Oracle® Retail Store Inventory Management

Operations Guide

Release 13.2.6

E41825-03

May 2016

ORACLE®

Oracle Retail Store Inventory Management Operations Guide, Release 13.2.6

E41825-03

Primary Author: Bernadette Goodman

reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

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

This documentation is in preproduction status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Contents

4 Backend System Configuration

List of Tables

List of Figures

Send Us Your Comments

Oracle Retail Store Inventory Management Operations Guide, Release 13.2.6

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

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

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

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

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

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

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

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

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

Preface

This Operations Guide provides critical information about the processing and operating details of Oracle Retail Store Inventory Management, including the following:

- System configuration settings
- Technical architecture
- Functional integration dataflow across the enterprise
- Batch processing

Audience

This guide is for:

- Systems administration and operations personnel
- Systems analysts
- Integrators and implementers
- Business analysts who need information about Oracle Retail Store Inventory Management processes and interfaces

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the *Oracle Retail Store Inventory Management Operaitons Guide* Release 13.2.6 documentation set:

- *Oracle Retail Store Inventory Management Release Notes*
- *Oracle Retail Store Inventory Management Installation Guide*

- *Oracle Retail Store Inventory Management Data Model*
- *Oracle Retail Store Inventory Management Operations Guide*
- *Oracle Retail Merchandising Batch Schedule*
- *Oracle Retail Merchandising Implementation Guide, volume 1 - Configuration*
- *Oracle Retail Store Inventory Management User Guide/Online Help*

Customer Support

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

https://support.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 13.2) or a later patch release (for example, 13.2.6). If you are installing the base release and additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

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

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1
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.

SIM 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 does 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.

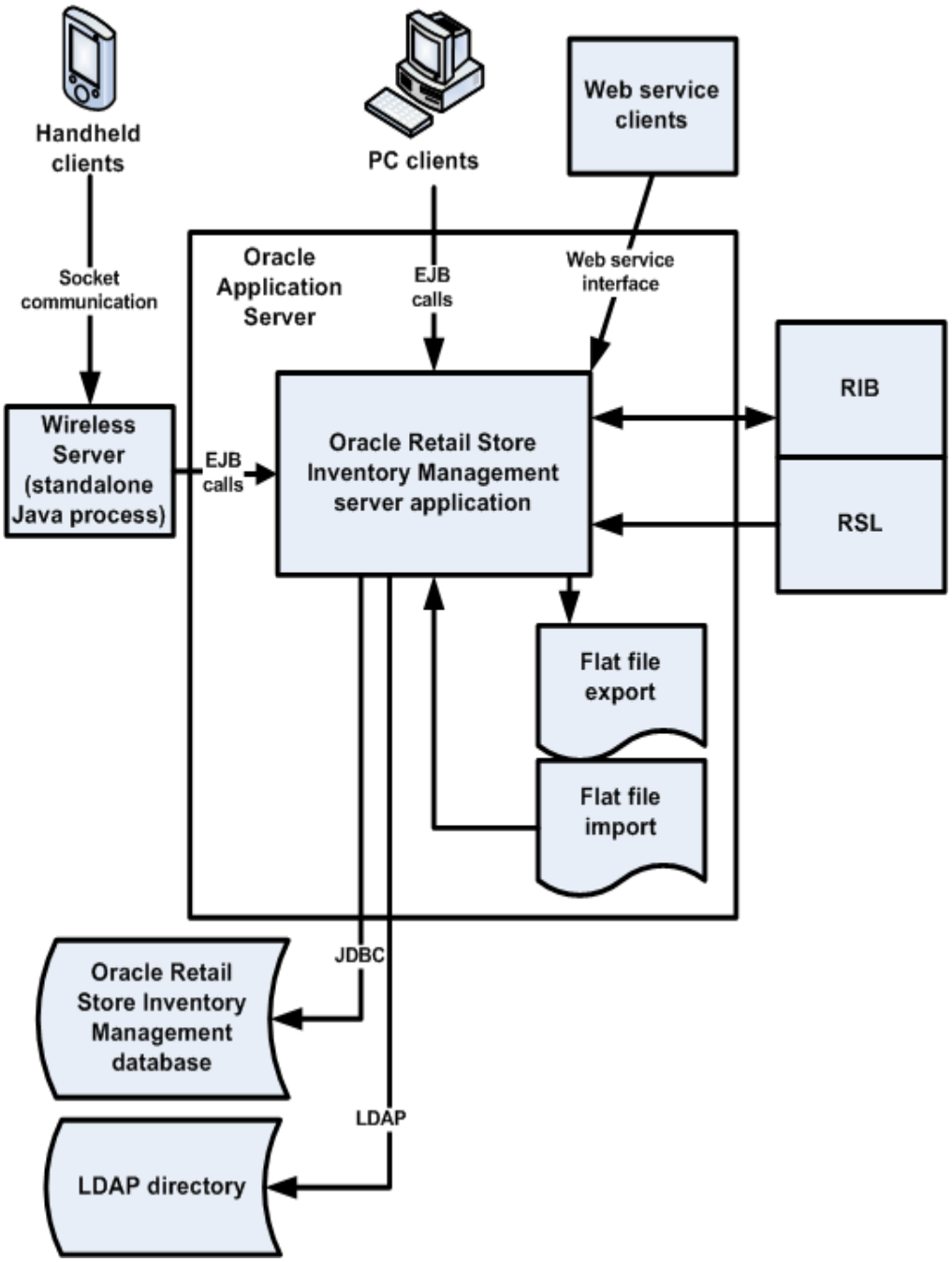
SIM has a client tier, a server tier, and a data tier. The n-tier architecture of SIM allows for the encapsulation of business logic, shielding the client from the complexity of the backend system. The separation of presentation, business logic, and data makes the software cleaner, more maintainable, 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 make direct updates to the server across the wire.

Figure 1–1 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 Chapter 2, "Technical Architecture".

Figure 1–1 SIM Technical Architecture



2

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.

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 Weblogic Application Server) and the Wavelink server (a standalone server for the handheld devices). The data tier consists of an Oracle 11g database and an LDAP directory.

The SIM login module makes use of the Weblogic security providers. The JNDI authentication happens with the user credentials to authenticate the user identity even before the SIM server application sets in, thus enhancing the security.

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:

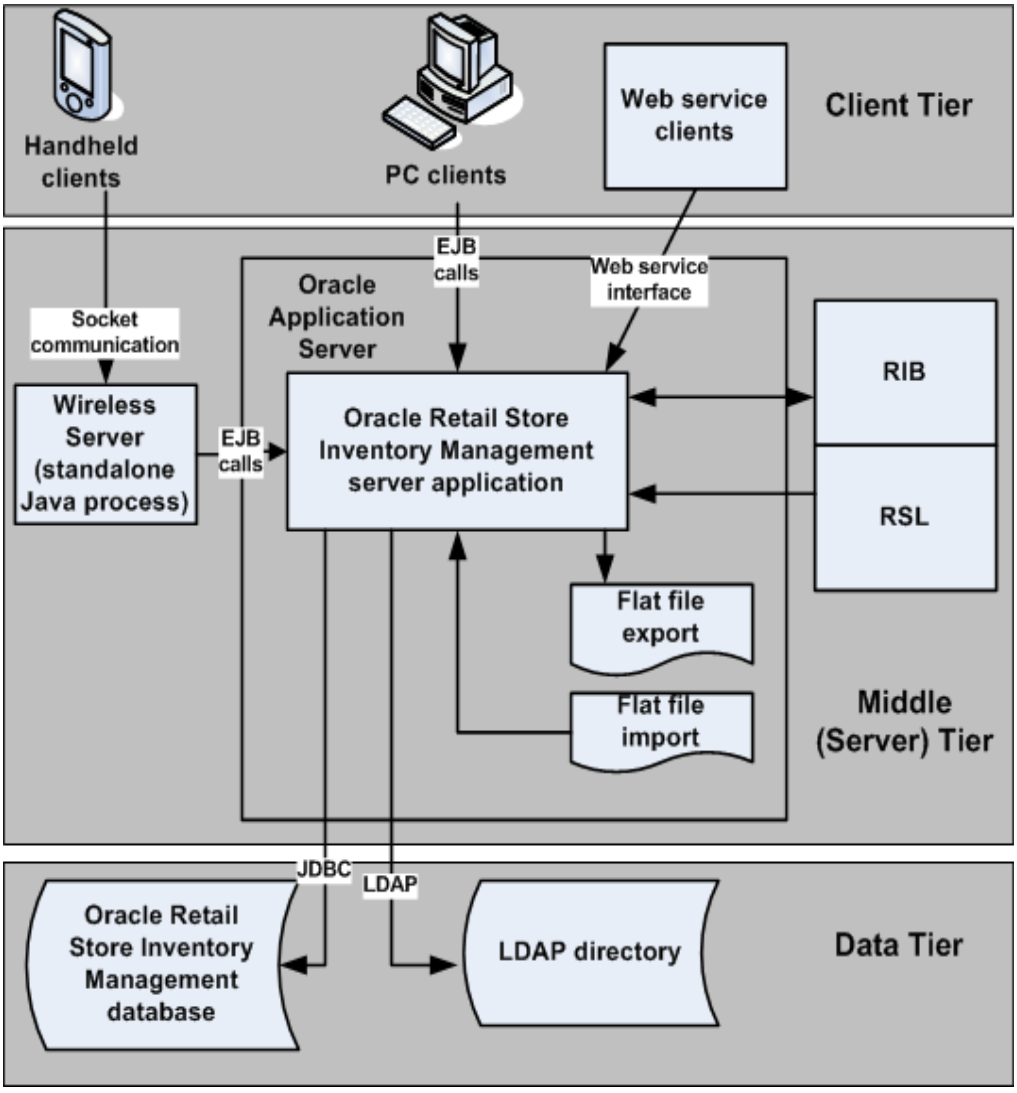
- Scalability: Hardware and software can be added to meet retailer requirements for each of the tiers.
- Maintainability: The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- Platform independence: The code is written once but 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. Figure 2–1 illustrates the major pieces of the typical three-tiered SIM implementation. Descriptions follow the diagram.

Figure 2–1 SIM Technical Architecture, Tiers



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 through 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 (APIs) 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.

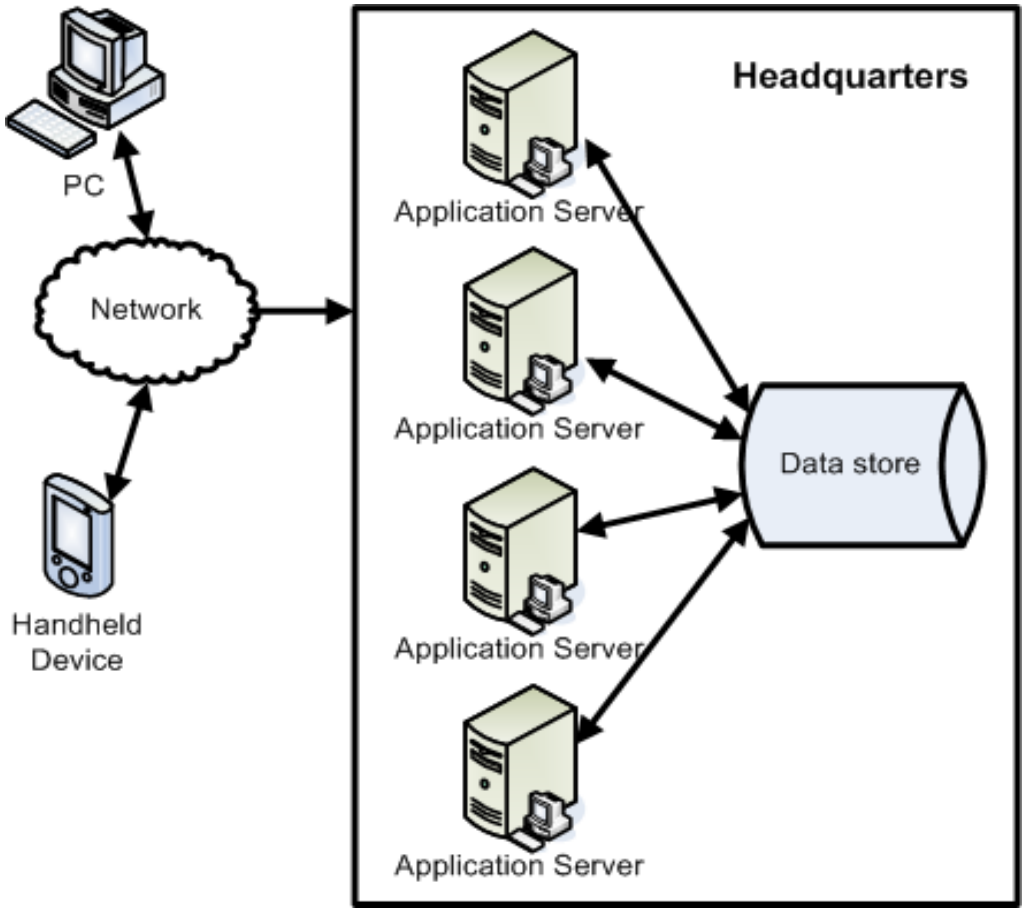
For example, if a helper class is called by the client to perform the method update shipment, the helper class appears to have that capability, though in reality it only behaves as a passage to the EJB remote reference, which is looked up from the server. The EJB remote reference communicates across 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 through 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.

Figure 2–2 illustrates SIM's deployment.

Figure 2–2 SIM Deployment



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.

3

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 run or scheduled through executable shell scripts (.sh files). Oracle Retail provides shell scripts for each SIM batch program.

The SIM batch program location is referred to as sim-batch-dir for the remainder of this chapter.

See "SIM Batch Scripts" in the *Oracle Retail Store Inventory Management Installation Guide* for batch install locations.

Each batch script performs the following internally:

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

Do the following to configure a batch environment:

1. Batch user logs in as valid batch user to the machine where SIM batch scripts are installed. The batch must have permission to execute the SIM shell script.

2. Set JAVA_HOME environment variable and add $JAVA_HOME/bin in the PATH environment variable. For example:

```
JAVA_HOME=<jre location>
PATH=$JAVA_HOME/bin:$PATH
export PATH JAVA_HOME
```

> **Note:** This command can be saved in a .profile file; the batch user can execute .profile before running SIM batches.

3. Execute the batch script from <*sim-batch-dir*>/bin.

For more information about batch usage, see "Batch Details" in this chapter.

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.

The user running the batch process must have write permission on the directory into which the sim batch log is written, or the batch process will not run. If it is not acceptable to give the batch user permission for the default log directory, log4j.xml must be configured to use a different directory.

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.

Summary of SIM Batch List

Table 3–1 summarizes SIM's batch programs and includes a description of each batch program's business functionality.

Table 3–1 Batch Process Business Functionality and Dependencies

Batch Program	Description	Dependencies
ActivatePriceChanges.sh	This batch process activates price changes which are effective today or on the user specified date.	No dependencies
AutoReceiveFinisherDeliveries	Auto-receives finisher deliveries.	No dependencies
AutoReceiveTransfers.sh	The batch process auto-receives transfers.	No dependencies
AutoReceiveWarehouseDeliveries.sh	The batch process auto-receives warehouse deliveries.	No dependencies
CleanupPickList.sh	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
ClearancePriceChange.sh	This batch process imports the clearance price changes set up in a price management system. SIM uses this data to update the price information of the items.	No dependencies
CloseProdGroupSchedule.sh	This batch process closes the product group schedule.	No dependencies
DeactivateOldUsers.sh	This batch program deactivates users when their end dates have reached specified date.	No dependencies
DexnexFileParser.sh	This batch imports the direct delivery shipment records (PO, shipment and receipt) from Dex/Nex files.	No dependencies
ExtractUnitAmountStockCount.sh	This batch program generates UnitAmount stock counts.	No dependencies
ExtractUnitStockCount.sh	This batch generates Unit stock counts.	No dependencies
ItemRequest.sh	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
ItemTicket.sh	The batch process generates item tickets/shelf labels for items with QR type images.	No dependencies
LateSalesInventoryAdjustmentPublishJob.sh	LateSalesInventoryAdjustmentPublishJob process publishes the late sale inventory adjustments records to Retail Merchandise System (RMS) through the Retail Integration Bus (RIB).	No dependencies
ProblemLineStockCount.sh	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, and so forth). The system automatically creates a stock count from those items that do fall within the parameters.	No dependencies
ProcessErrorPosTransactions.sh	This batch reprocess the pos transactions which are in error state.	No dependencies
ProcessStagedMessages.sh	This batch allows for a remote process to kick off server side polling timer threads on demand.	No dependencies
PromotionPriceChange.sh	This batch process imports the promotional price changes setup in a price management system. SIM uses this data to update the price information of the items.	No dependencies
PurgeAdHocStockCount.sh	This batch process deletes ad hoc stock counts with a status of **in progress**.	No dependencies

Table 3–1 (Cont.) Batch Process Business Functionality and Dependencies

Batch Program	Description	Dependencies
PurgeAll.sh	This process deletes records from the SIM application that meet certain business criteria.	No dependencies
PurgeAudits.sh	This batch process deletes audits.	No dependencies
PurgeCompletedUINDetail.sh	This batch deletes completed UIN Detail records.	No dependencies
PurgeCustomerOrders.sh	This batch deletes closed customer order records.	No dependencies
PurgeDeletedUsers.sh	This batch program deletes users that have a status of deleted and have been held in the system for a number of system defined days.	No dependencies
PurgeDSDReceivings.sh	This batch process deletes the Direct Store Delivery receiving.	No dependencies
PurgeInvalidUserRoles.sh	This batch program removes all expired or orphaned roles from the users in the system.	No dependencies
PurgeInventoryAdjustments.sh	This batch process deletes inventory adjustments.	No dependencies
PurgeItem.sh	This batch program deletes the items that are in deleted status (D)	No dependencies
PurgeItemBaskets.sh	This batch process deletes item baskets	No dependencies
PurgeItemRequests.sh	This batch process deletes item requests.	No dependencies
PurgeItemTickets.sh	This batch process deletes item tickets.	No dependencies
PurgeLockings.sh	This batch process deletes lockings.	No dependencies
PurgePickList.sh	This batch process deletes pick lists.	No dependencies
PurgePosTransactionHistory	This batch program deletes the POS_ TRANSACTION_HISTORY records which are at least *X* days in the past.	No dependencies
PurgePriceChanges.sh	This batch process deletes price changes.	No dependencies
PurgePriceHistories.sh	This batch process deletes price histories.	No dependencies
PurgePurchaseOrders.sh	This batch process deletes purchase order records.	No dependencies
PurgeReceivedTransfers.sh	This batch process deletes received transfers.	No dependencies
PurgeResolvedUINProblems.sh	This batch process deletes resolved item serial number problems.	No dependencies
PurgeSalesPosting.sh	This purges the sales, returns, void sales, and void returns transaction from the staging table.	No dependencies
PurgeStagedMessage.sh	This batch process removes processed integration staging records.	No dependencies
PurgeStockCounts.sh	This batch process deletes stock counts.	No dependencies
PurgeStockReturns.sh	This batch process deletes stock returns.	No dependencies
PurgeTemporaryUINDetail.sh	This batch process deletes temporary UIN detail records.	No dependencies
PurgeUINDetailHistories.sh	This batch process deletes UIN detail history records (UIN Audit information).	No dependencies
PurgeUserCache.sh	This batch program deletes expired user cache records.	No dependencies
PurgeUserPasswordHistory.sh	This batch program deletes user password history records.	No dependencies

Table 3–1 (Cont.) Batch Process Business Functionality and Dependencies

Batch Program	Description	Dependencies
PurgeWHDReceivings.sh	This batch process deletes the Warehouse delivery receivings.	No dependencies
RegularPriceChange.sh	This batch process imports the permanent/regular price changes setup in a price management system. SIM uses this data to update price information of the items.	No dependencies
ResaCustomerOrderFileParser.sh	This batch imports the Inventory Reservation information related to a Customer Order from ReSA.	No dependencies
ResaFileParser.sh	This batch process imports sales and returns data that originates in Oracle Retail Point-of-Service. SIM uses the data to update the SOH for the store/items combinations in each file.	No dependencies
ReturnNotAfterDateAlert.sh	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 through the system's administration GUI screens.	No dependencies
ThirdPartyStockCountParser.sh	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
TransfersOverdueBatch.sh	This batch process sends user e-mail for dispatched transfers which have not been received after a number of days.	No dependencies
WastageInventoryAdjustments.sh	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.sh	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 Notes

Most SIM batches can be scheduled to run at any time (ad hoc) with no particular order, while some of batches might provide optimal results when batches are run in a particular order. Table 3–2 provides some scheduling recommendations:

Table 3–2 Batch Process Scheduling Notes

Batch Name	Schedule Type	Successor Depends on Success of Predecessor	Notes
ResaFileParser.sh LateSalesInventoryAdjustmentPublishJob.sh	Daily/ad hoc	Not required. The successor batch runs regardless of success/failure of the predecessor batch.	These batches should ideally be run together. Running them together will increase inventory accuracy.
DeactivateOldUsers.sh PurgeDeletedUsers.sh PurgeUserPasswordHistory.sh PurgeInvalidUserRoles.sh PurgeUserCache.sh PurgeItem.sh	Ad hoc	Not required. The successor batch runs regardless of success/failure of the predecessor batch.	These batches should run on a continuous basis to ensure tight security and appropriate access to SIM.
CleanupPickList.sh	Daily		This batch should be run at least once a day if the delivery bay is used.
WastageInventoryAdjustments.sh WastageInventoryAdjustmentPublishJob.sh	Daily	Not required. The successor batch runs regardless of success/failure of the predecessor batch.	These batches should not run ad hoc as they will decrement inventory. They should run one time a day and in combination to ensure better inventory accuracy.
AutoReceiveFinisherDeliveries.sh AutoReceiveTransfers.sh AutoReceiveWarehouseDeliveries.sh	Daily/ad hoc	Not required. The successor batch runs regardless of success/failure of the predecessor batch.	These batches should be run at least once per day or for appropriate receipt closures.

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

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.

Note: The Activate Price Change batch can extract time-based promotions. The batch picks the new price based on the date and time input and activates the price. The Activate Price change batch allows the user to input the time optionally along with the date on which the batch is run. The date and time format is **dd/MM/yyyy HH:mm.**

This batch is backward-compatible with the dd/MM/yyyy format. Only the promotion part is modified in Activate Price Change batch to handle time-based promotions.

Usage

The following command runs the ActivatePriceChanges batch job:

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

Where the `activate_date` is optional, date format must be `dd/MM/yyyy HH:mm` 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.

Note: The price effective date in SIM database is stored as GMT date.

When integrating with a price management application for pricing information, the **Enable GMT for Price Changes** system option must always be set to **no** since the pricing date from a price management application is not a GMT date.

If a user opts for time-based promotions, set up a cron job to run ActivatePriceChange.sh every 30 minutes in order to activate or deactivate price changes:

1. Enter the `crontab -e` command.
2. Append the following:

```
15 * * * * /{Local sim batch
path}/bin/ActivatePriceChanges.sh
```

AutoReceiveFinisherDeliveries Batch

AutoReceiveFinisherDeliveries does the following:

- Retrieves a list of all stores.
- For each store, if the External Finisher Auto Receive store parameter is set to **Date Driven**, then the batch auto-receives all finisher deliveries that are in **New** status and whose Estimated Time of Arrival (ETA) added to the **External Finisher Auto Receive Number Of Days** is less than the current date.

Usage

The following command runs the AutoReceiveFinisherDeliveries batch:

```
AutoReceiveFinisherDeliveries.bat
```

AutoReceiveTransfers

AutoReceiveTransfers does the following:

- Retrieves a list of all stores.
- For each store, if the Store Auto Receive store parameter is set to **Date Driven** and the receiving-from store is set up for auto-receiving, then the batch auto-receives all transfers that are either in Dispatched status at the sending store and In Transit status at receiving store, and whose Ship Date added to the Store Auto Receive Number Of Days store parameter value is less than or equal to the batch date.

- The batch will not auto receive the transfers in following scenarios:
  - UIN processing is enabled at Receiving Store and the transfer has UIN enabled items.
  - The transfer is locked by any other user or program.

  In these scenarios, the batch will skip the transfer and will continue with the next record.

Usage

The following command runs the AutoReceiveTransfers batch:

```
AutoReceiveTransfers.sh <ship_date>
```

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

AutoReceiveWarehouseDeliveries

AutoReceiveWarehouseDeliveries does the following:

- Retrieves a list of all stores.
- For each store, if the Warehouse Auto Receive store parameter is set to **Date Driven**, then the batch auto-receives all warehouse deliveries that are in New status and whose Estimated Time of Arrival (ETA) added to the Warehouse Auto Receive Number Of Days store parameter value is less than or equal to the batch date.
- The batch will not auto receive the warehouse deliveries in following scenarios:
  - UIN processing is enabled at Store and the delivery has UIN enabled items.
  - The delivery is locked by any other user/program.

  In these mentioned scenarios, the batch will skip the warehouse delivery and will continue with the next record.

Usage

The following command runs the AutoReceiveWarehouseDeliveries batch:

```
AutoReceiveWarehouseDeliveries.sh <ship_date>
```

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

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. 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

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.

DeactivateOldUsers Batch

This batch process finds active users that have passed their end date and updates their status in table AC_USER to inactive.

Usage

The following command runs the DeactivateOldUsers batch:

```
DeactivateOldUsers.sh <purge_date>
```

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

DexnexFileParser Batch

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
```

The parameters DEXNEX_INPUT_DIR and DEXNEX_ERROR_DIR are present in the rk_config table which can be accessed through the admin console in the SIM GUI, and config value can be entered through it or can be edited directly in the database.

ExtractUnitAmountStockCount Batch

This batch program generates UnitAmount 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. For example, if a scheduled count includes a list of five stores, then five separate stock count records are created.

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.

Usage

The following command runs the ExtractUnitAmountStockCount batch:

```
ExtractUnitAmountStockCount.sh <extract_date>
```

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

ExtractUnitStockCount Batch

This batch program generates Unit 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.For example, if a scheduled count includes a list of five stores, then five separate stock count records are created.

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.

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

Usage

The following command runs the ExtractUnitStockCount batch:

```
ExtractUnitStockCount.sh <extract_date>
```

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

ItemRequest

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 POs or warehouse to store transfers as applicable.

Usage

The following command runs the ItemRequest batch:

```
ItemRequest.sh
```

ItemTicket

ItemTicket batch looks at the item/image table for any QR type item that has a date equal to the date for which the batch is running. The batch creates the item tickets or shelf labels provided the store options are set and no open ticket or label exists.

The batch uses the following store options:

- Send item tickets to ticketing for QR code changes.
- Send item labels to ticketing for QR code changes.

Usage

The following command runs the ItemTicket batch:

```
ItemTicket.sh <batch_date>
```

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

LateSalesInventoryAdjustmentPublishJob

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.

This batch can be run anytime. For example, it can be run after ResaFileParser batch for each store or run after ResaFileParser batch completes for all stores.

This batch program is not designed to run in multiple thread mode. Doing so could lead to duplicate inventory adjustments for late sales transactions.

Usage

The following command runs the LateSalesInventoryAdjustmentPublishJob.sh:

```
LateSalesInventoryAdjustmentPublishJob.sh
```

ProblemLineStockCount Batch

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
```

ProcessErrorPosTransactions Batch

This batch processes the error pos transactions. With the direct updates Web service integration between Oracle Retail Point-of-Service and SIM, the sale, return, void of sale and void of return get directly updated to SIM. The inventory updates take place in SIM. However during the processing of these transactions, if there is an error because of incorrect data and it is corrected by the user, the ProcessErrorPosTransactions batch can be run to process the particular record or set of records.

The batch queries the staging table where all the Point-of-Service transactions are persisted (POS_TRANSACTION) and checks for the three flags.

Usage

The following command runs the ProcessErrorPosTransactions batch:

```
ProcessErrorPosTransactions.sh
```

After the batch process is complete, the POS_TRANSACTION table can be checked for the following flags, to check if the processing of each one of them took place correctly:

- SALE_RET_TXN_PROCESS_SUCCESS
- INV_RESV_PROCESS_SUCCESS
- UIN_PROCESS_SUCCESS

If all the three flags are updated to **Y**, then all the transactions have been processed successfully.

If the soh update is not turned on, then the flag SALE_RET_TXN_PROCESS_SUCCESS will be set to **null**. Only if the flag is turned on, will it be set to **Y** or **N**, based on the fact that the transaction processing was successful or failed. The soh update flag is sent by Oracle Retail Point-of-Service in the request while invoking the Web service.

Similar logic applies to the other two flags present in the POS_TRANSACTION table.

ProcessStagedMessages Batch

This batch allows for a remote process to kick off server side polling timer threads on demand. This can enable a customer to kick off threads to have burst message processing during certain times of the day, or allow for more control in scheduling the polling timer threads outside of the internal polling time subsystem that runs within the JEE app server (which is controlled by the polling coordinator).

Usage

```
<threads> [<max_messages_per_thread> <iterations>]
```

Where:

- `threads`: defines how many threads to spawn.

  No default value.

- `max_messages_per_thread`: defines the maximum number of messages each thread can have.

  Default **30**.

- `iterations`: defines how many times the staged message table is checked for more messages to process (a value of **-1** causes the batch to process until it cannot find anymore messages).

  Default **1**.

For example:

- ProcessStagedMessage.sh 10
- ProcessStagedMessage.sh 10 20
- ProcessStagedMessage.sh 10 40 2
- ProcessStagedMessage.sh 10 60 -1

ResaFileParser Batch

This batch program imports sales transaction data (POSU file) that originated in Oracle Retail Point-of-Service. The external audit system will provide in its sales upload file a percentage or quantity that indicates how much the inventory needs to be reduced by, in addition to the sold quantity.

For example, meat will become lighter as fluids evaporate. Other items, for example cheese or ham, will only be reduced when of the outside layers are cut off to sell the item.

SIM takes the sales transaction data to update the store item's inventory buckets. From the batch program, SIM learns about inventory movement (that is, what is sold and what is returned). Once SIM attains the data, SIM assumes that sales should be taken from the store's shelf-related inventory buckets. This assumption is important to SIM's shelf replenishment processing. Similarly, SIM assumes that returns should go to the backroom bucket; the system's logic is that returns must be inspected.

The batch also writes each processed records into a transaction history table. See PurgePosTransactionHistory for details.

Usage

From SIM batch location, run command

```
ResaFileParser.sh <filename1> [<filename2> <filename3>]
```

Where `filename` (required): the name of the input POSU file which contains the sales transaction data from one store.

ResaFileParser batch has the ability to process multiple files. File names are separated by a space.

ResaFileParser batch commits transactions based on the commit maximum setting. It can be configured through COMMIT_MAX_CTR in the RK_RESTART_CONTROL table.

File Location Permission

- The ReSA database directory objects must exist in SIM database. The ReSA database directory objects and corresponding operating system directories are created during SIM database installation.

  The actual file location on the database server can be found by executing the following queries:

  ```
  Select directory_name, directory_path from all_directories where directory_name
  in ('RESA_DIR', 'RESA_ORIGINAL_DIR', 'RESA_LOG_DIR');
  ```

- The corresponding operating system directories for the ReSA data files must be created on the SIM database server.

- The Oracle process invoked by the SIM application user must have full access to the directory locations specified by RESA_DIR, RESA_ORIGINAL_DIR and RESA_LOG_DIR. The read and write privileges on these directory objects must be granted to SIM schema owners and SIM schema users.
- The system administrator must ensure that the operation system directory has the correct read and write permissions for the Oracle database processes.
- The ReSA data file must reside on the database server.
- The minimum required permissions for the input file should be given:

```
rw_rw_r__
```

> **Note:** Minimal permissions of RW to the group users are required to allow Oracle Process to move the file to archive directory. SIM schema user must belong to the group user for ReSA operation system directories.

Batch Detail

ResaFileParser takes the sales transaction data and updates the store item's inventory buckets (store item's total quantity, shop floor quantity, and so forth) if applicable.

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

- If an item in the file has an item level below the transaction level (for example, item level = 3, transaction level = 2) and not ranged (exist) in the store, then it is an invalid record, it will be written to the rerun file.
- If an item in the ReSA file has an item level equal to the transaction level and is not ranged (exist) in the store, then a new ranged item record will be created for the item/store, and store item's inventory buckets will be updated.

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

- If an item in file has an item level below the transaction level (for example, item level= 3, transaction level = 2), then batch process will compare the parent item's item level and transaction level as following:
  - If ref item's parent item level equals the transaction level, and parent item exists in the store, then the stock on hand of the parent item will be updated.
  - If ref item's parent item is a transaction level item, but is not ranged (exists) for the store, then a new ranged item is created for that store, and the stock on hand for the parent item will be updated.

For late sale items:

- A late sale is a sales transaction occurring before a stock count is completed, and the sale data file is processed after the count has started.
- SIM system parameter stock_count_sales_processing with value of Timestamp Processing or Daily Sales Processing indicates if the POSU sales transaction data contains the transaction date timestamp.
- 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 sales transaction is a late sale.
- Daily Sales Processing indicates sale data in the POSU 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 also used to determine whether the stock count occurred before or after business hours so that SIM knows how to handle late sales.

- When SIM encounters a ReSA late-sale item, it must correct the inventory for stock counts that have been processed after the time of the sale. The stock count creates inventory adjustments for discrepancies during the count that are out of sync. The ResaFileParser will attempt to correct the inventory buckets by creating an inventory adjustment with the reversal in the amount indicated in the ReSA flat sale file.
- The batch process records the late sales records in POS Transaction History table as Non-Inventory impact transaction.

For open stock count items:

- An open stock count item is the in-progress stock count item while ResaFileParser is running.
- For open stock count items, in addition to updating the store item's inventory buckets, the batch refreshes the stock count snapshot if applicable (for example, the snapshots are refreshed for stock counts which are not authorized).

Restart and Recover

ResaFileParser batch commits transactions based on the commit maximum counter. It can be configured through COMMIT_MAX_CTR in the RK_RESTART_CONTROL table.

Once an error in the processing is resolved, ResaFileParser.sh batch may be restarted/recovered from the point of failure.

The following tables are used in batch restart process:

- RK_RESTART_CONTROL
- RK_RESTART_BOOKMARK
- RK_RESTART_PROGRAM_STATUS
- RK_RESTART_PROGRAM_HIST

For each file, ResaFileParser batch spawns a separate thread that processes the file. Each thread records the starting of the process in RK_RESTART_PROGRAM_STATUS table with the file name as the RESTART_NAME. Once the process completes successfully, the record is deleted from the program status table; a history record is then inserted into the RK_RESTART_PROGRAM_HIST table for each completed process.

The NON_FATAL_ERR_FLAG in RK_RESTART_PROGRAM_HIST table indicates whether the particular thread or file is completed with non-fatal error (for example, batch processes encountered an invalid record, rerun file is created with the invalid records).

While a thread processes a file, if the thread fails on a fatal error, then the thread marks the process as **ABORTED** in the RK_RESTART_PROGRAM_STATUS table, with the file name as the RESTART_NAME.

The thread stores the last commit point in the RK_RESTART_BOOKMARK table, with the file name as the RESTART_NAME.

The system administrator can view the error and make corrections if applicable. Once the file is ready to run again, the administrator sets the RESTART_FLAG to **Y** in the

RK_RESTART_PROGRAM_STATUS table. In the event of a restart, the process begins from the last commit point.

Once the error is corrected, and the RESTART_FLAG in RK_RESTART_PROGRAM_STATUS table is set to **Y** by the user, the user can start running ResaFileParser batch for that particular file.

The system administrator needs to purge RK_RESTART_PROGRAM_HIST records periodically based on their requirements.

Multi-Threading

When multiple files are sent to ResaFileParser batch for processing, the batch spawns multi-threads based on the number of threads configured in the restart control table. Each file can only be processed by one thread; the same data file will never be acted upon by multiple processes.

The number of parallel threads to execute the batch processes can be configured. To configure the thread numbers for the batch, update the NUM_THREADS column in the RK_RESTART_CONTROL table for program_name RESA_FILE_PARSER.

Error Handling, Logging and File Archiving

ResaFileParser batch writes the logging information and invalid records into log file; the batch also creates rerun files for invalid records, and the uncommitted records in the event of a failure. System administrator needs to purge the archive files and log files periodically based on their requirements.

Archive file: When the batch completes processing a file successfully, the batch moves the ReSA input data file to an archive location which is specified by the RESA_ORIGINAL_DIR database directory object.

Rerun file: If the ReSA batch program encounters invalid records or a failure occurs during the process, the batch creates a re-run file. This file contains all invalid or uncommitted records from the original process. The re-run file is located in the same directory as the file being processing (specified by the RESA_INPUT_DIR database configuration value).

Once the error is corrected and all the invalid records are corrected (the invalid records are also logged in the log file), the rerun file can be sent to ResaFileParser batch for processing. The rerun file has the following naming convention:

`<resa_file>_rerun<YYYYMMDDHH24:MI:SS>`

Log file: ResaFileParser batch writes the log file to the location specified by RESA_LOG_DIR database directory object. The log file has the following naming convention:

`<resa_file>.log`

ReturnNotAfterDateAlert Batch

This batch process warns users a number of days in advance that the RTV/RTW is about to reach the **Not After** date and must be dispatched. The value for the number of days of advance warning is configurable using the system's administration screens.

Usage

The following command runs the ReturnNotAfterDateAlert batch:

```
ReturnNotAfterDateAlert.sh
```

StockCountAuthorizeRecovery Batch

This batch process looks for stock counts that are stuck in Authorize Processing state. This is a unique state that appears when an error occurs during the final processing of a stock count. The batch attempts to fully authorize the stock count. Errors that occur during the batch process are logged to the server error logs and will indicate the reason for any further processing failures. 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

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.

The stock count in the input data file must exist in SIM. See Appendix: Creating an Auto-Authorized Third-Party Stock Count for information about setting up third-party stock count in SIM.

Batch Design Overview

Non-auto authorized stock count is a stock count which is generated from a product group which has auto-authorized flag turned off. Manual authorize stock count is needed after ThirdPartyStockCount batch completes:

- 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 external stock counting system, the stock count records type is set to **authorize** and the status is set to **in progress**. The user must manually authorize the stock after the import process completes.
- If any items are sent from external stock counting 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 an external stock counting system to SIM, any unknown item data is written to the Rejected table.
- Any not-on-file or not-at-store items can be assigned a valid SIM item ID on SIM Rejected Items screen.

Auto-authorized stock count is a stock count which is generated from a product group which has auto-authorized flag turned on. The stock count is automatically authorized by the ThirdPartyStockCount batch:

- 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 external stock counting system, the stock count records type is set to **authorize** and the status is set to **completed**.

- If any items are sent from external stock counting system 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 external stock counting to SIM, any unknown item data is written to the Rejected table.
- The authorization process occurs as part of the import of the third party file. Note that in this case, any items that are considered Rejected or Not at Store items cannot be assigned to a valid SIM item ID. The auto-authorization 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 and generates an export file to RMS with the same stock count data.
- The user does not need to manually authorize the stock after the import process completes.
- Once the import process is complete, SIM automatically authorizes the unit and amount stock counts and exports the stock count data to RMS. The location of this export upload file to RMS is specified by the database directory object STOCK_COUNT_UPLOAD_DIR. Under normal operating circumstances, this manual process is triggered by a SIM user through the SIM PC Stock Count Screen.

ThirdPartyStockCount Integration Assumptions

- RMS provides an item export file to external stock counting prior to the count in order for external stock counting to validate the items that are scanned.
- The items coming from external stock counting 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 external stock counting 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 external stock counting 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 external stock counting 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 stock counting count of 0, the discrepancy check uses the variance units (not the variance percentage) value to determine whether the item is discrepant, user can view the discrepant items through Stock Count Rejected Items PC screen after batch completes.

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 must be granted to the schema owner and schema user. The stock count import data file must 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.

By default, when the batch process completes, the input data file and log file stay at the same location as specified by STOCK_COUNT_DIR directory object.

Optionally, the input data files and log files can be moved to a different location after batch completes.

Do the following to move/archive input files to a different location:

1. DBA creates STOCK_COUNT_ARCHIVE_DIR database directory object, which points to the physical directory in the operating system that can be accessed by Oracle process invoked by SIM application schema user.
2. Grant read/write privileges to SIM application schema user.

Do the following to move/archive log files to a different location:

1. DBA creates STOCK_COUNT_LOG_DIR database directory object, which points to the physical directory in the operating system that can be accessed by Oracle process invoked by SIM application schema user.
2. Grant read/write privileges to SIM application schema user.

The following command runs ThirdPartyStockCountParser batch:

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

Where:

- `file_name` is the import file data from one store; the stock count import data file must be put at the location specified by the STOCK_COUNT_DIR Oracle directory. 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 775 to change the stock count import data file before start the batch.
- `snapshot` is an optional argument that indicates if batch will automatically take the snapshot for the stock count if snapshot has not been taken. Snapshot is required prior to authorizing the stock count.

TransfersOverdue Batch

This batch process sends user e-mail for dispatched transfers which have not been received after a number of days. The value for the number of days of e-mail alert is configurable using the system's administration screen.

Usage

The following command runs the TransfersOverdue batch:

```
TransfersOverdueBatch.sh
```

WastageInventoryAdjustments Batch

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.

Note: Wastage is not run for items that require UINs.

Usage

Following command runs the WastageInventoryAdjustments batch:

```
WastageInventoryAdjustments.sh
```

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

WastageInventoryAdjustmentPublishJob Batch

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 Process

Transactional and historical records in SIM can be purged as follows:

- 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 for each batch.

PurgeAdHocStockCount Batch

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
```

PurgeAll Batch

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:

- Deleted items
- Received transfers
- Stock Counts
- Inventory Adjustments
- Warehouse Receivings
- DSD/DSDASN Receivings
- Stock Returns
- Price Changes
- Price Histories
- Pick Lists
- Item Requests
- Customer Order
- Item Tickets
- Audits
- Sales Posting
- Lockings
- Adhoc Stock Counts
- Integration Staging
- Temporary UINs
- Resolved UIN Problem
- Completed UINs
- UIN detail histories
- Purchase Order

Usage

`PurgeAll.sh <purge_date>`

Where `purge_date` is optional, date format must be `dd/MM/yyyy` if purge_date is specified.

PurgeAudits

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 dd/MM/yyyy if purge_date is specified.

PurgeCompletedUINDetail

This batch program deletes completed UIN Detail records. A completed UIN is any UIN with a status of Removed from Inventory, Missing, Sold, Shipped to Vendor or Shipped to Warehouse. Any UIN detail record with a complete status and update date at least *X* days in the past (where *X* is with system parameter DAYS_TO_HOLD_COMPLETED_UINS) will be deleted from UIN_DETAIL table.

Usage

```
PurgeCompletedUINDetail.sh <purge_date>
```

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

PurgeCustomerOrders

This batch finds customer records with status not in Open status and update date is at least X days in the past (where X is the system parameter DAYS_TO_HOLD_COMPLETED_CUSTOMER_ORDERS), and deletes these records from CUSTOMER_ORDER_LINE_ITEM_UIN, CUSTOMER_LINE_ITEM and CUSTOMER_ORDER tables.

Usage

```
PurgeCustomerOrders.sh <purge_date>
```

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

PurgeDeletedUsers

This batch process finds users marked as deleted with an end date that is at least *X* days in the past (where *X* is the system parameter SECURITY_DAYS_TO_HOLD_DELETED_USERS). These users and all associated data are deleted from AC_USER_ROLE, AC_USER_STORE, AC_USER_PASSWORD and AC_USER tables.

Usage

```
PurgeDeletedUsers.sh <purge_date>
```

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

PurgeDSDreceivings Batch

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 DSDASNs 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 `dd/MM/yyyy` if purge_date is specified.

PurgeInvalidUserRoles

This batch program removes all expired user roles and orphaned user roles (roles that were deleted by removing a store) from the SIM system.

The batch process finds user role assignments that have an end date that is at least *X* days in the past, where *X* is the system parameter SECURITY_DAYS_TO_HOLD_EXPIRED_USER_ROLES, and deletes these expired role assignments. The users (excluding super users) with role assignments that have no matching store assignments (orphaned role assignments) are also deleted from AC_USER_ROLE table.

Usage

```
PurgeInvalidUserRoles.sh <purge_date>
```

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

PurgeInventoryAdjustments Batch

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 `dd/MM/yyyy` if purge_date is specified.

PurgeItem Batch

This batch program deletes items with a status of Delete (D).

There are two segments in the PurgeItem Batch which do the following different tasks:

- Validate if the Item should be deleted.
- Delete item from all associated tables if validation check is passed.
- Validate if the item should be deleted. The Validations include:
  - If SOH of item, item parent and item grandparent is 0
  - If any transfers exist for item, item parent and item grandparent
  - If any RTV exists for item, item parent and item grandparent
  - If any Inventory adjustment exists for item, item parent and item grandparent and so on.
- Delete item from all associated table. If the validations checks are met, the records related to the item which is marked for the purge action are deleted.

Usage

```
PurgeItem.sh
```

PurgeItemBaskets Batch

This batch process deletes item baskets. Any item basket record with a process date or timestamp older than batch date value is deleted.

Usage

```
PurgeItemBaskets.sh <purge_date>
```

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

PurgeItemRequests Batch

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 `dd/MM/yyyy` if purge_date is specified.

PurgeItemTickets Batch

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 dd/MM/yyyy if purge_date is specified.

PurgeLockings Batch

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 dd/MM/yyyy if purge_date is specified.

PurgePickList Batch

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 dd/MM/yyyy if purge_date is specified.

PurgePosTransactionHistory

This batch program deletes the POS_TRANSACTION_HISTORY records which are at least *X* days in the past.

Usage

```
PurgePosTransactionHistoryJob.sh <number_of_days>
```

Where number_of_days is optional and is the number of days in the past. A default of seven days is used if no other value is specified.

PurgePriceChanges Batch

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 <purge_date>
```

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

PurgePriceHistories Batch

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 <purge_date>
```

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

PurgePurchaseOrders

This batch process deletes closed purchase order records. The purchase order records which are in closed status and complete date is at least *X* days in the past (where *X* is system parameter DAYS_TO_HOLD_COMPLETED_PURCHASE_ORDERS) are deleted from the database.

Usage

```
PurgePurchaseOrders.sh <purge_date>
```

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

PurgeReceivedTransfers Batch

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 `dd/MM/yyyy` if purge_date is specified.

PurgeResolvedUINProblems Batch

This batch process deletes resolved UIN exception records. UIN exception records with status of resolved and resolved date is at least *X* days in the past (where *X* is system parameter DAYS_TO_HOLD_RESOLVED_UIN_EXCEPTIONS) are deleted from UIN_PROBLEM DETAIL table.

Usage

```
purgeResolvedUINProblems.sh <purge_date>
```

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

PurgeSalesPosting Batch

This batch process deletes the Point-of-Service transaction from the Oracle Retail Point-of-Service transaction staging table. It reads the Days to Hold Sales Posting configuration parameter and all the transactions which are present beyond the configuration parameter are deleted.

Usage

```
PurgeSalesPosting.sh <purge_date>
```

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

PurgeStagedMessage Batch

This batch finds integration staging records that are marked as processed or deleted, and update date is at least *X* days in the past (where *X* is the system parameter DAYS_TO_HOLD_COMPLETED_STAGING_RECORDS), the batch process deletes these records from STAGED_MESSAGE table.

Rebuilding the indexes on the STAGED_MESSAGE table each day is recommended after batch process completes.

Usage

```
PurgeStagedMessageJob.sh <purge_date>
```

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

Note: The optional date is the point of reference the script should use. The script uses current date for the point of reference by default, but can be run from any reference point.

Therefore, if purging records that are 10 days old, and running the script without the optional date, the process removes all records older than 10 days from the current date. If the optional date argument is specified, records 10 days older than the specified optional date are purged.

PurgeStockCounts Batch

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 `dd/MM/yyyy` if purge_date is specified.

PurgeStockReturns Batch

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 `dd/MM/yyyy` if purge_date is specified.

PurgeTemporaryUINDetail

This batch process deletes temporary UIN detail records. UIN detail records with no status and update date is at least *X* days in the past (where *X* is system parameter DAYS_TO_HOLD_TEMPORARY_UINS) are deleted from UIN_DETAIL table.

Usage

```
PurgeTemporaryUINDetail.sh <purge_date>
```

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

PurgeUINDetailHistories

This batch process deletes UIN detail history records (UIN Audit Information). UIN Audit information could be purged for a UIN while the UIN is still open within the system. Open UIN is any UIN that is in one of the following statuses:

- In Stock
- In Receiving
- Reserved for Shipping
- Unavailable

UIN history records with open status and an update date at least *X* days in the past (where *X* is system parameter DAYS_TO_HOLD_UIN_AUDIT_INFORMATION) are deleted from HST_UIN_DETAIL table.

Usage

```
purgeUINDetailHistories.sh <purge_date>
```

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

PurgeUserCache Batch

This batch program deletes expired user cache records which include expired user authentication caches and authorization cache records.

The cached user passwords with a cache date that is at least *X* hours in the past (where *X* is the system parameter SECURITY_USER_AUTHENTICATION_CACHE_HOURS) will be deleted.

The batch process also find users with a cache date that is at least *X* hours in the past (where *X* is the system parameter SECURITY_USER_AUTHORIZATION_CACHE_HOURS). All cached role and store assignments are deleted for these users.

If the users still have existing role assignments, store assignments, or passwords then the user's cache date is set to **null**.

If the user has no existing assignments or passwords then the user is deleted from AC_USER_PASSWORD, AC_USER_ROLE, AC_USER_STORE and AC_USER tables.

Usage

```
PurgeUserCache.sh <purge_date>
```

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

If purge_date is not specified, the current GMT time is used.

PurgeUserPasswordHistory Batch

This batch process finds users with more than *X* passwords (where *X* is the system parameter PASSWORD_NUMBER_OF_PREVIOUS_TO_DISALLOW), and deletes the oldest passwords that exceed this limit.

Usage

```
PurgeUserPasswordHistory.sh <purge_date>
```

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

Purge_date currently has no effect, reserved for future use.

PurgeWHDReceivings Batch

This batch process deletes the Warehouse delivery receivings which are in Completed or 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 `dd/MM/yyyy` if purge_date is specified.

Price Bulk Processing Batch Process

This appendix provides the details for executing Price Bulk Processing (PBP) batch programs in SIM. Price Bulk Processing imports the price changes from a price management system 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.

> **Note:** The Price Bulk Processing batch can accept an input file with a start date and end date with a time component for each promotion record. The file layout incorporates Start_date and End_date.
>
> Start_date and End_date can accept the date and time in a **yyyymmddhh24miss** format. This batch is backward compatible with the yyyymmdd format. Only the promotion part is modified in Price Bulk Processing batch to handle time-based promotions.

Usage

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 script defines the main price change files and their descriptions:

PromotionPriceChange.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 a price management application 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.

- The 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 deletes price changes for the items/locations combination from the RK_PRICE_CHANGE table for which no data exists in the flat file for active promotions, and the corresponding item tickets of the deleted price changes that are generated for active promotion will also be deleted from RK_ITEM_TICKET table.
- BulkPricePromotionJob sets the thread IDs to a logical unit of promotion information based on the number of threads configured for the 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.

ClearancePriceChange.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 a price management application in BPP_INPUT_DIR directory.
2. Run 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.

- The 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.

RegularPriceChange.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 a price management application in BPP_INPUT_DIR directory
2. Run 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.

- The 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.

TimeStamp.log

- <<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.

Restart and Recovery

Each of the three batch processes in Price Bulk Processing are multi-threaded. These processes have the ability to recover from the point of failure of each thread. While processing records, if a particular thread fails, then the batch writes a record for this thread in the table RK_RESTART_PROGRAM_STATUS with the following values in the column:

- ERR_MESG: the error message due to which the thread failed.
- program_status: ABORTED
- RESTART_FLAG: **N**

The last commit point for this thread is saved in the table RK_RESTART_BOOKMARK. The system administrator can view the error and make any correction if needed. Once the records are ready to run again, the administrator sets the RESTART_FLAG to **Y** in RK_RESTART_PROGRAM_STATUS table. In the event of a restart, the process begins from the last commit point.

ResaCustomerOrderFileParser Batch

This batch imports the Inventory Reservation information related to a Customer Order from ReSA. The Customer Order transactions originate from Oracle Retail Point-of-Service as Customer Pickup, Layaway, Customer Order and Pending Purchase transactions.

In Oracle Retail Point-of-Service, the customer reserves the inventory of an item for later pickup. These items have to be reserved in SIM and inventory of these items should me moved to the Unavailable bucket. In SIM, the reserved inventory is accounted in Customer Order Quantity bucket and this bucket is considered as unavailable inventory. The available Stock On Hand is updated accordingly.

When the items are delivered as part of pickup in Oracle Retail Point-of-Service, the Customer Order Quantity bucket has to be updated to un-reserve the inventory.

The ResaCustomerOrderFileParser performs the following:

- Import the inventory reservation information from the flat file into the staging table.
- Identifies bad records and discarded records, and logs them in a separate file. The bad and discarded records are identified by running the business rules on the input details. The business rules include:
  - Items and Store combination exists in SIM.
  - Checks for the valid UOM of the item.
  - The items must be transaction level or lower in order for the system to process.
  - Consignment and non-sellable items are not considered for Inventory Adjustment.
  - Quantity should be greater than zero.
  - The final quantity of the Customer Order Quantity after reserving and unreserving should not be negative.
  - Validate the action for each item. The valid actions are:
    * New
    * CancelReservation
    * Fulfill
    * CancelFulfill
  - Identify duplicate entries of Customer Order.
- Read the Customer Order header and detail information from the staging table. The header information includes the Store ID, Customer Order number, and Timestamp. The detail information includes Item Number, Quantity and Action.
- Based on action for each item, determine if the final quantity for the item is not negative. For example, in a Customer Order, if the reservation quantity of an item is 10, the fulfilled quantity cannot be more than 10. If the reservation quantity is more than 10, it violates the business rule that the user had picked up more quantity than the reserved quantity.
- Update the Customer Order tables for each item and quantity
- If the item is other than Consignment and Concession Item and non-sellable item, create Inventory Adjustment using the reason codes as
  - Customer Order Reservations – In
  - Customer Order Reservations - Out
- In a customer order, when the reservation quantity of an item is equal to the fulfilled quantity, then the customer order is completed.

Assumptions

If the UOM is not passed as input, the UOM of the item is assumed to be Standard UOM.

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

- The input Customer Order file has to be placed in RESA_CO_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 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.

Do the following to run the script:

1. Place the Customer Order input files from ReSA in RESA_CO_DIR directory.
2. Run the ResaCustomerOrderFileParser.sh at the command prompt with Customer Order input files as arguments separated by a space.

Usage

`ResaCustomerOrderFileParser.sh <filename1> <filename2>`

Where `filename1` and `filename2` are separated by a space. The order for processing the files is from left to right.

If multiple input files are considered for processing, each file will be processed by a separate thread. The batch supports up to a max number of threads configured for this process.

The 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 ResaCustomerOrderFileParserJob Java class in SIM server.

ResaCustomerOrderFileParserJob calls the ResaCustOrderFileParserProcedure Java. The ResaCustOrderFileParserProcedure invokes start_cust_order_resa_parser stored procedure in RESA_CUST_ORDER_FILE_PARSER package. This stored procedure loads the data from the customer order input file into the staging tables in SIM and processes it. After the successful processing, it moves the input file to the RESA_CO_ARCHIVE_DIR archive directory.

The log file for each execution is generated and placed in the RESA_CO_LOG_DIR directory.

This batch also supports Restart and Recovery process.

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 broken 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 subsequently listed. They generally work in the following pattern:

1. Query the server to find a set of data that needs to be processed.
2. Break the set of data into units of work that can be worked on independently in separate threads.
3. Create threads to work concurrently on the units of work.
4. Wait for all threads to finish.
5. 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.cfg (located at sim-home/files/prod/retek/sim.cfg). If no value is specified, a default value of 4 is used.

Batch Programs that Create Threads

- Price Bulk Processing Batches
- ResaFileParser

4

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:

- Enable GMT for Dex/Nex
- Enable GMT for Direct Deliveries
- Enable GMT for Foundation Data
- Enable GMT for Inventory Adjustment
- Enable GMT for Item Requests
- Enable GMT for Price Changes
- Enable GMT for Receiving
- Enable GMT for RTVs
- Enable GMT for Sales Data
- Enable GMT for Stock Counts
- Enable GMT for Store Orders
- Enable GMT for Store Transfers
- Enable GMT for Third Party Stock Count
- Enable GMT for Vendor ASN
- Enable GMT for Warehouse Transfers

Note: When integrating with a price management application for pricing information, the **Enable GMT for Price Changes** option must always be set to **no** because a price management application is not a 24/7 system.

- 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, 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

For information about integration compatibility for this release and for requirements for SIM's client, servers, and database, see the *Oracle Retail Store Inventory Management 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.

SIM configuration files are packaged as various resources jars, this section list the most commonly used configuration files.

The sim13appplication.zip contains:

sim/application/sim13/sim-client/sim-client.ear

- sim-client.war

lib/sim-client-resources.jar:,

log4j.xml

conf/date.cfg

conf/client.cfg

lib/sim-common-resources.jar

conf/common.cfg

sim/application/sim13/sim-server/sim-server.ear

lib/sim-server-resources.jar:

log4j.xml

conf/server.cfg

conf/rules_sim.xml

conf/ldap.cfg

conf/jndi.cfg

lib/sim-common-resources.jar:,

common.cfg

lib/sim-int-rib13_2-resources.jar

lib/(or sim-int-rib13_1-resources.jar):

sim/application/sim13/wireless.zip

resources/log4j.xml

resources/conf/wireless.cfg

resources/conf/rules_sim.xml

resources/conf/jndi.cfg

resources/conf/date.cfg

resources/conf/common.cfg

sim/application/sim13/sim-batch.zip

resources/log4j.xml

resources/conf/jndi.cfg

resources/conf/common.cfg

sim/application/sim13/sim-int/sim-int-rib13_2-resources.jar

retek/services_rsl.xml

retek/service_flavors.xml

retek/jndi_providers.xml

retail/retail_service_config_info_ribclient.xml

retail/remote_service_locator_info_ribclient.xml

retail/injectors.xml

Note: The sim-int resources jar various based on the RIB versions with which SIM integrates. The sim-int-rib13_1-resources.jar file is for SIM integrating with RIB 13.1.x, and the sim-int-rib13_2-resources.jar file is for SIM integrating with RIB 13.2.

sim/application/sim13/sim-int/sim-int-rib13_1-resources.jar

retek/services_ribclient.xml

retek/service_flavors_ribclient.xml

retek/service_context_factory.xml

retek/service_context_factory.xml

retek/services_rib.xml

retek/services_rms.xml

retek/services_rpm.xml

retek/rib/injectors.xml

Note: Within these files (and thus in some of the examples from those following files), 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.

After making configuration file changes, the updated .ear files must be redeployed.

client.cfg

The client.cfg file contains the following:

- STARTUP_DISPLAY – this class must be StartupDisplayer or a sub-class of StartupDisplayer.
- INITIALIZERS – comma-delimited class name list that is executed upon SIM PC client startup.
- NATIVE.COMMANDS – comma-delimited list of native commands that display in stats area.
- CLIENT_LOCK_PORT – the port used to lock the client application.
- SHUTDOWN – shutdown command line.
- Screen components used by application main frame:
  - GUI.APPFOLDER
  - GUI.MAINFRAME
  - GUI.GLOBALBAR
  - GUI.APPTOOLBAR
  - GUI.LOGGERFACTORY
  - GUI.STATUSDISPLAYER
  - GUI.DEFAULT_SCREEN
  - GUI.EXIT
    - NORMAL
    - SPIN
    - DISSOLVE
    - SHRINK
- REPORTING_SERVICE_BROWSER_LAUNCHER –Reports Browser launcher.

- REPORTS_EXECUTABLE – this value holds the executable for displaying the reports portal.
- HELP_EXECUTABLE – this value holds the executable for displaying help.

Example 4–1 Example client.cfg File

```
# This class must be StartupDisplayer or a sub-class of StartupDisplayer
STARTUP_DISPLAY=oracle.retail.sim.closed.application.StartupDisplayer

# A comma delimited class name list that is executed upon SIM PC client startup
# Each entry must be an implementation of
oracle.retail.sim.closed.common.Initializer.
INITIALIZERS=oracle.retail.sim.closed.bootstrap.SimClientInitializer

# Comma delimited list of native commands that will show up in stats area.
NATIVE.COMMANDS=

# The port which is used to lock the client application.  This port will be
checked on each client
# invocation to make sure only one instance of this application is running.
CLIENT_LOCK_PORT=51803

# Shutdown command line
SHUTDOWN=RESTART_CLIENT

# Screen components used by application main frame
GUI.APPFOLDER=sim
GUI.MAINFRAME=oracle.retail.sim.closed.swing.sim.SimApplicationFrame
GUI.GLOBALBAR=oracle.retail.sim.shared.swing.core.SimStatusBar
GUI.APPTOOLBAR=oracle.retail.sim.shared.swing.core.SimToolbar
GUI.LOGGERFACTORY=oracle.retail.sim.shared.swing.core.SimLoggerFactory
GUI.STATUSDISPLAYER=oracle.retail.sim.shared.swing.core.SimStatusDisplayer
GUI.DEFAULT_SCREEN=oracle.retail.sim.shared.swing.login.MainScreen

# Screen Exit Options are NORMAL, SPIN, DISSOLVE, SHRINK
GUI.EXIT=NORMAL

# Reports Browser launcher
REPORTING_SERVICE_BROWSER_
LAUNCHER=oracle.retail.sim.shared.report.launcher.bipublisher.BIPublisherBrowserRe
portLauncher

# This value holds the executable for displaying the reports portal.
# Need a different executable for Linux
REPORTS_EXECUTABLE=cmd /c start {0}

# This value holds the executable for displaying help.
# Available parameters: {0} - the absolute path of the URL which should display
#                             the help documentation.
HELP_EXECUTABLE=cmd /c start {0}
```

common.cfg

The following keys define the implementation to classes that instantiate objects in SIM through the factory pattern:

- BO_FACTORY_IMPL
- WSO_FACTORY_IMPL

- DEO_FACTORY_IMPL
- CLIENT_COMMAND_FACTORY_IMPL
- CLIENT_SERVICE_FACTORY_IMPL
- SERVER_SERVICE_FACTORY_IMPL

Where:

- BO – Business Objects
- WSO – Web Service Objects
- DEO – Data Exchange Objects

The common.cfg file also includes:

- CURRENCY_DEFAULT_TYPE – default currency type for non-specified currency.
- STOCK_COUNT_MAX_AUTH_LINES – indicates the maximum number of lines that can be handled by the authorization process at one time.
- BATCH_NUM_THREADS_IN_POOL – the number of threads that execute concurrently to get batch work done when SimBatch.executeBatchCallables is used.
- RESA_CO_FILE_INPUT_DIR – specifies the directory object on the database for ReSA customer order data file.
- RESA_CO_FILE_ARCHIVE_DIR – specifies the directory object on the database for archiving ReSA original customer order input file.
- RESA_CO_LOG_DIR – specifies database directory object for the ReSA customer order log file and rerun file, if exists.
- MASK_FACTORIES – the following mask factories determine the implementation responsible for instantiating the phone and money masks (parsing/formatting):
  - PHONE_MASK_FACTORY
  - MONEY_MASK_FACTORY
- Cache Refresh Rates of Commonly Cached Information.

  Value is in milliseconds:
  - 30000=30 seconds
  - 300000=5 minutes
  - 3600000=1 hour

  The default setting is 3600000.
- REFRESH_RATE_ACTIVE_INV_ADJ_REASON – reason codes for inventory adjustments cached on handheld server (milliseconds).
- REFRESH_RATE_CONFIG – system configuration values cached on handheld server, main server and PC client (milliseconds).
- REFRESH_RATE_DELIVERY_TIMESLOT – delivery timeslots cached on handheld server (milliseconds).
- REFRESH_RATE_FINISHER_RETURN_REASON – return reason code for a return to finisher cached on the handheld server (milliseconds).
- REFRESH_RATE_AGSN_TICKET_FORMAT – AGSN Item Ticket formats cached on the handheld server (milliseconds).

- REFRESH_RATE_ITEM_TICKET_FORMAT – Regular Item Ticket Formats cached on the handheld server (milliseconds).
- REFRESH_RATE_MERCH_HIERARCHY – item merchandise hierarchy (depts) cached on handheld server, main server and PC client (milliseconds).
- REFRESH_RATE_PRICE_HISTORY – price history cached on handheld server (milliseconds).
- REFRESH_RATE_SHELF_LABEL_FORMAT – shelf label formats cached on the handheld server (milliseconds).
- REFRESH_RATE_STORE – store configuration cached on handheld server, main server and PC client (milliseconds)
- REFRESH_RATE_SUPPLIER – suppliers cached on handheld server (milliseconds).
- REFRESH_RATE_SUPPLIER_RETURN_REASON – return reason codes for supplier returns on handheld server (milliseconds).
- REFRESH_RATE_TRANSLATION – translations cached on handheld server and main server (milliseconds).
- REFRESH_RATE_UIN_CONTAINER – UIN Container details cached on handheld server (milliseconds).
- REFRESH_RATE_WAREHOUSE – warehouses cached on handheld server (milliseconds).
- REFRESH_RATE_WAREHOUSE_RETURN_REASON – return reason codes for warehouse returns on handheld server (milliseconds).
- REFRESH_RATE_WIRELESS_ITEM – differentiator information cached on handheld server (milliseconds).

Example 4–2 Example common.cfg File

```
# These keys define the implementation to classes that instantiate objects in SIM
via the factory pattern
# BO = Business Objects, WSO = Web Service Objects, DEO = Data Exchange Objects
BO_FACTORY_IMPL=oracle.retail.sim.shared.business.BOFactoryImpl
WSO_FACTORY_IMPL=oracle.retail.sim.shared.business.WSOFactoryImpl
DEO_FACTORY_IMPL=oracle.retail.sim.shared.integration.deo.DEOFactoryImpl
CLIENT_COMMAND_FACTORY_
IMPL=oracle.retail.sim.shared.business.ClientCommandFactoryImpl
CLIENT_SERVICE_FACTORY_
IMPL=oracle.retail.sim.shared.business.ClientServiceFactoryImpl
SERVER_SERVICE_FACTORY_
IMPL=oracle.retail.sim.shared.business.ServerServiceFactoryImpl

# Default currency type for non-specified currency
CURRENCY_DEFAULT_TYPE=USD

# Indicates the maximum number of lines that will be handled by authorization
process at one time.
# In other words, authorization of stock count will loop on total line items and
only process 5000
# line items within each loop.
STOCK_COUNT_MAX_AUTH_LINES=5000

# The number of threads that execute concurrently to get batch work done when
SimBatch.executeBatchCallables is used.
```

```
BATCH_NUM_THREADS_IN_POOL=5

# RESA_CO_FILE_INPUT_DIR specifies the directory object on database for ReSa
customer order data file.
# RESA_CO_FILE_ARCHIVE_DIR specifies the directory object on database for
archiving ReSa original customer order input file.
# RESA_CO_LOG_DIR specifies database directory object for the ReSa customer order
log file and rerun file if exist.
#
# These database directory objects are created by DBA at SIM installation time.
# The actual file directory on database server can be found by following query:
#     select directory_name, directory_path from dba_directories where directory_
name in
#     ('RESA_CO_DIR', 'RESA_CO_ARCHIVE_DIR', 'RESA_CO_LOG_DIR')
RESA_CO_FILE_INPUT_DIR=RESA_CO_DIR
RESA_CO_FILE_ARCHIVE_DIR=RESA_CO_ARCHIVE_DIR
RESA_CO_LOG_DIR=RESA_CO_LOG_DIR

# These mask factories determine the implementation responsible for instantiating
the phone and money masks (parsing/formatting).
PHONE_MASK_FACTORY=oracle.retail.sim.shared.swing.format.PhoneMaskFactoryImpl
MONEY_MASK_FACTORY=oracle.retail.sim.shared.swing.format.MoneyMaskFactoryImpl

# Cache Refresh Rates Of Commonly Cached Information.
# Value is in milliseconds. 30000=30 seconds  300000=5 minutes  3600000=1 hour
# The default setting is 1 hour unless configured here

# Reason Codes for inventory adjustments cached on handheld server (milliseconds)
REFRESH_RATE_ACTIVE_INV_ADJ_REASON=30000

# System configuration values cached on handheld server, main server and PC client
(milliseconds)
REFRESH_RATE_CONFIG=30000

# Delivery timeslots cached on handheld server (milliseconds)
REFRESH_RATE_DELIVERY_TIMESLOT=30000

# Return reason code  for a return to finisher cached on the handheld server
(milliseconds)
REFRESH_RATE_FINISHER_RETURN_REASON=30000

# AGSN Item Ticket formats cached on the handheld server (milliseconds)
REFRESH_RATE_AGSN_TICKET_FORMAT=30000

# Regular Item Ticket Formats cached on the handheld server (milliseconds)
REFRESH_RATE_ITEM_TICKET_FORMAT=30000

# Item merchandise hierarchy (depts) cached on handheld server, main server and PC
client (milliseconds)
REFRESH_RATE_MERCH_HIERARCHY=30000

# Price history cached on handheld server (milliseconds)
REFRESH_RATE_PRICE_HISTORY=30000

# Shelf label formats cached on the handheld server (milliseconds)
REFRESH_RATE_SHELF_LABEL_FORMAT=30000

# Store configuration cached on handheld server, main server and PC client
(milliseconds)
REFRESH_RATE_STORE=30000
```

```
# Suppliers cached on handheld server (milliseconds)
REFRESH_RATE_SUPPLIER=30000

# Return reason codes for supplier returns on handheld server (milliseconds)
REFRESH_RATE_SUPPLIER_RETURN_REASON=30000

# Translations cached on handheld server and main server (milliseconds)
REFRESH_RATE_TRANSLATION=30000

# UIN Container details cached on handheld server (milliseconds)
REFRESH_RATE_UIN_CONTAINER=30000

# Warehouses cached on handheld server (milliseconds)
REFRESH_RATE_WAREHOUSE=30000

# Return reason codes for warehouse returns on handheld server (milliseconds)
REFRESH_RATE_WAREHOUSE_RETURN_REASON=30000

# Differentiator information cached on handheld server (milliseconds)
REFRESH_RATE_WIRELESS_ITEM=30000
```

date.cfg

This file defines the 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 where xxXX is the two-letter ISO language code plus country code. Both language and country must be present. Additional language/country combinations can be added as desired. 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 month and day only
- wirelessInput–defines entry for wireless device
- wirelessOutput–defines the format of dates on the wireless device
- wirelessDisplay–defines the exact text string to display to the user at the entry location
- firstDayOfWeek – the first day of the week to display on calendar pop-up

Editing date.cfg

The configuration file is located at `files/prod/config/date.cfg`.

The file must be changed and then placed in the classpath.

Valid days of week values are:

- 1 = Sunday
- 2 = Monday
- 3 = Tuesday
- 4 = Wednesday
- 5 = Thursday
- 6 = Friday
- 7 = Saturday

The enUS represents the language (en=english) and country (US=United States) of the user logged in. There are examples for many other countries already provided in the date.cfg file.

Example 4–3 Example date.cfg File

```
# ENGLISH - UNITED STATES
#enUS.firstDayOfWeek=1
#enUS.entryDate=M/d/yy
enUS.shortDate=M/d/yyyy
#enUS.mediumDate=MMM d, yyyy
#enUS.longDate=MMMM d, yyyy
#enUS.fullDate=EEEE, MMMM d, yyyy
enUS.monthPattern=MM-dd
enUS.wirelessInput=MM-dd-yy,MMddyy
enUS.wirelessOutput=MM-dd-yy
enUS.wirelessDisplay=mm-dd-yy
```

where:

- The pound sign (#) indicates the option is commented out and that the default JAVA format will be used for that option.

jndi.cfg

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

- INITIAL_CONTEXT_FACTORY – the name of the factory used to get an initial JNDI context. 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.
- OBJECT_FACTORY_PACKAGES – the Java packages containing object factories. This should not be changed.
- SECURITY_PRINCIPAL – the user name to connect to the Weblogic Application Server's JNDI context. The SIM installer should have set this.
- SECURITY_CREDENTIALS – the password to connect to the Weblogic Application Server's JNDI context. The SIM installer should have set this.
- EJB_MAX_CONNECT_ATTEMPTS – the maximum number of times to try to connect to the EJB server. Should always be at least **2** to allow for server unexpected reconnects.

Example 4–4 Example jndi.cfg File

```
# The InitialContextFactory class (Required)
INITIAL_CONTEXT_FACTORY=weblogic.jndi.WLInitialContextFactory
```

```
# The URL for the naming server (Required)
NAMING_SERVER_URL=t3://localhost:7003

# The Java packages (colon-delimited) containing object factories (Required)
OBJECT_FACTORY_PACKAGES=weblogic.jndi

# App server username/password
SECURITY_PRINCIPAL=weblogic
SECURITY_CREDENTIALS=weblogic1

# The maximum number of times to try to connect to the EJB server.
# Should always be at least 2 to allow for server unexpected reconnects.
EJB_MAX_CONNECT_ATTEMPTS=2
```

Note: For SIM deployed on OAS server, need to use OAS-specific settings, for example:

- INITIAL_CONTEXT_ FACTORY=oracle.j2ee.rmi.RMIInitialContextFactory
- OBJECT_FACTORY_PACKAGES =oracle.aurora.jndi
- NAMING_SERVER_ URL=ormi://${appserver.host}:${appserver.rmi.port}/${deploy.app.name}
- SECURITY_PRINCIPAL=oc4jadmin
- SECURITY_CREDENTIALS=oc4jadmin

ldap.cfg

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

Example 4–5 Example ldap.cfg File

```
#
# LDAP Configuration file
#

#URL and port number for the ldap server
#PRIMARY_LDAP_URL=ldap://<LDAPServerName or IP Address>:389
PRIMARY_LDAP_URL=@input.ldap.url@
#BACKUP_LDAP_URL_1=@input.ldap.url@
#BACKUP_LDAP_URL_2=@input.ldap.url@

#LDAP Schema Details
BASE_DN=@input.ldap.base.dn@
APPLICATION_LOGIN=@input.ldap.user@
APPLICATION_PASSWORD=@input.ldap.password@

#Initial context factory to use.
#Is the value for javax.naming.Context.INITIAL_CONTEXT_FACTORY
CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory

#Security
#Specifies the authentication mechanism that the Service Provider will use.
Permitted values:
```

```
#none - use no authentication (anonymous)
#simple - use weak authentication (clear text password)
#Should not be modified in most cases.
SECURITY_LEVEL=simple

#Connection pooling (com.sun.jndi.ldap.connect.pool)
CONNECT_POOL=true
```

server.cfg

The server.cfg file contains the following:

- INITIALIZERS – comma-delimited class name list that must be executed when the SIM server starts.
- FINALIZERS – comma-delimited class name list that must be executed when the SIM server stops.
- DB_JNDI_NAME – the name the application's DataSource is registered under.
- MAIL_JNDI_NAME – the name the application's MailSession is registered under.
- DB_LOCK_WAIT_TIME – time in seconds to wait on database locks.
- DB_BATCH_MAX_PARAM – maximum number of parameters allowed in a database batch statement.
- The following keys define the implementation to classes that instantiate objects in SIM through the factory pattern:
  - DAO_FACTORY_IMPL
  - CACHE_DAO_FACTORY_IMPL
  - RSL_DAO_FACTORY_IMPL
  - COMMAND_FACTORY_IMPL
  - WEB_COMMAND_FACTORY_IMPL
- The following factories look up appropriate consumers, publishers, stagers, and handlers (for staged message processing):
  - MESSAGE_CONSUMER_FACTORY_IMPL
  - MESSAGE_PUBLISHER_FACTORY_IMPL
  - MESSAGE_STAGER_FACTORY_IMPL
  - MESSAGE_HANDLER_FACTORY_IMPL
- Server cache refresh rates. Time is in milliseconds (30000=30 seconds, 300000=5 minutes):
  - REFRESH_RATE_USER_AUTHENTICATION_CACHE
  - REFRESH_RATE_USER_AUTHORIZATION_CACHE
- USER_PASSWORD_ALGORITHM – algorithm used to encrypt user passwords that are stored in SIM.
- REPORTING_SERVICE_REQUEST_ADAPTER – key for Reporting service request adapter
- POLLING_TIMERS_ENABLED – enable or disable the entire polling timer sub-system (true | false).

- POLLING_TIMERS_STATS_ENABLED – enable or disable the polling timer stats (true | false).
- POLLING_TIMERS_PROCESS_OUTBOUND_MESSAGES – enable or disable the polling timers from processing outbound messages (true | false).
- POLLING_TIMERS_PROCESS_INBOUND_MESSAGES – enable or disable the polling timers from processing inbound messages (true | false).
- POLLING_TIMERS_MAX_CONCURRENT_THREADS – the maximum number of concurrent polling timers threads that might be running (integer)
- POLLING_TIMERS_MAX_MESSAGES_PER_THREAD – the maximum number of messages a single polling timer thread might process at once (integer)
- POLLING_TIMERS_MAX_MESSAGE_RETRIES – the maximum number of times a message might be retried before it is considered failed (integer).

> **Note:** This number can be fairly high to allow the retry polling timers sufficient attempts.

- POLLING_TIMERS_COORDINATOR_TIMEOUT_SECONDS – the amount of time the polling coordinator sleeps before checking on the status of messages and threads in the system (integer).

> **Note:** This is a lightweight thread that can be run every 5 to 10 seconds with little impact on server.

Example 4–6 Example server.cfg File

```
# INITIALIZERS: A comma delimited class name list that needs to be executed when
the sim server starts.
# Each entry must implement oracle.retail.sim.closed.common.Initializer.
INITIALIZERS=oracle.retail.sim.closed.bootstrap.SimServerInitializer

# FINALIZERS: A comma delimited class name list that needs to be executed when the
sim server stops.
# Each entry must implement oracle.retail.sim.closed.common.Finalizer.
# For an example of what could be done, see
oracle.retail.sim.closed.bootstrap.TestFinalizer
FINALIZERS=oracle.retail.sim.closed.bootstrap.SimServerFinalizer

# The name the application's DataSource is registered under.
DB_JNDI_NAME=jdbc/SimDataSource

# The name the application's MailSession is registered under.
MAIL_JNDI_NAME=mail/SimMailSession

# Time in seconds to wait on database locks
DB_LOCK_WAIT_TIME=5

# Maximum number of parameters allowed in a database batch statement
DB_BATCH_MAX_PARAM=5000

# These keys define the implementation to classes that instantiate objects in SIM
via the factory pattern
DAO_FACTORY_IMPL=oracle.retail.sim.shared.dataaccess.DAOFactoryImpl
CACHE_DAO_FACTORY_IMPL=oracle.retail.sim.shared.dataaccess.CacheDAOFactoryImpl
```

```
RSL_DAO_FACTORY_IMPL=oracle.retail.sim_int_rib13_2.shared.rsl.RSLDAOFactoryImpl
COMMAND_FACTORY_IMPL=oracle.retail.sim.shared.business.CommandFactoryImpl
WEB_COMMAND_FACTORY_IMPL=oracle.retail.sim.shared.business.WebCommandFactoryImpl

# These factories lookup appropriate consumers, publishers, stagers, and handlers
(for staged message processing)
MESSAGE_CONSUMER_FACTORY_
IMPL=oracle.retail.sim.shared.integration.consumer.SimMessageConsumerFactoryImpl
MESSAGE_PUBLISHER_FACTORY_IMPL=oracle.retail.sim_int_rib13_
2.shared.publisher.SimMessagePublisherFactoryImpl
MESSAGE_STAGER_FACTORY_
IMPL=oracle.retail.sim.shared.integration.stager.SimMessageStagerFactoryImpl
MESSAGE_HANDLER_FACTORY_IMPL=oracle.retail.sim_int_rib13_
2.closed.SimMessageHandlerFactoryImpl

# Server cache refresh rates. Time is in milliseconds. 30000=30 seconds  300000=5
minutes
REFRESH_RATE_USER_AUTHENTICATION_CACHE=30000
REFRESH_RATE_USER_AUTHORIZATION_CACHE=30000

# Algorithm used to encrypt user passwords that are stored in SIM
USER_PASSWORD_ALGORITHM=SHA

# Key for Reporting service request adapter
REPORTING_SERVICE_REQUEST_
ADAPTER=oracle.retail.sim.shared.report.adapter.bipublisher.BIPublisherServiceRequ
estorAdapter

# Enable or disable the entire polling timer sub-system (true|false)
POLLING_TIMERS_ENABLED=true

# Enable or disable the polling timer stats (true|false)
POLLING_TIMERS_STATS_ENABLED=true

# Enable or disable the polling timers from processing outbound messages
(true|false)
POLLING_TIMERS_PROCESS_OUTBOUND_MESSAGES=true

# Enable or disable the polling timers from processing inbound messages
(true|false)
POLLING_TIMERS_PROCESS_INBOUND_MESSAGES=true

# The maximum number of concurrent polling timers threads that may be running
(integer)
POLLING_TIMERS_MAX_CONCURRENT_THREADS=10

# The maximum number of messages a single polling timer thread may process at once
(integer)
POLLING_TIMERS_MAX_MESSAGES_PER_THREAD=30

# The maximum number of times a message may be retried before it is considered
failed (integer)
# Note: This number can be fairly high to allow the retry polling timers
sufficient attempts
POLLING_TIMERS_MAX_MESSAGE_RETRIES=1000000

# The amount of time the polling coordinator sleeps before checking on the status
of messages and threads in the system (integer)
# Note: This is a lightweight thread that can be run every 5 to 10 seconds with
little impact on server
```

```
POLLING_TIMERS_COORDINATOR_TIMEOUT_SECONDS=5
```

wireless.cfg

This file contains configuration used by the Wireless Server:

- INITIALIZERS_CLIENT – comma-delimited class name list that is executed upon wireless client startup.
- INITIALIZERS_SERVER – comma-delimited class name list that is executed upon wireless server startup.
- Wireless Port – port the wireless server runs on, and clients connect to.

Example 4–7 Example wireless.cfg File

```
# INITIALIZERS_CLIENT: A comma delimited class name list that is executed upon
wireless client startup
# Each entry must be an implementation of
oracle.retail.sim.closed.common.Initializer.
INITIALIZERS_
CLIENT=oracle.retail.sim.closed.bootstrap.SimWirelessClientInitializer

# INITIALIZERS_SERVER: A comma delimited class name list that is executed upon
wireless server startup
# Each entry must be an implementation of
oracle.retail.sim.closed.common.Initializer.
INITIALIZERS_
SERVER=oracle.retail.sim.closed.bootstrap.SimWirelessServerInitializer

# Wireless Port - This is the port that the wireless server runs on, and clients
connect to.
PORT=40099
```

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:

- WLS HTTP port (to download the SIM client)
- WLS 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 *Oracle Retail Store Inventory Management Installation Guide* for more information.

The Weblogic 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:

- *Weblogic 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, and so forth). Only ten files are kept. If ten files already exist and the current file rolls over, the oldest log file is deleted.

Default Location of Log Files

The following describes the default location of the server log files and the client log files.

Server Log Files

The server log file location can be changed by changing the value of the File parameter in the sim.appender appender log4j.xml file. See Configuration Files for log4j.xml locations.

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 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

It is possible to change the level of any logger in the log4j.xml 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 the OC4J instance for SIM.
3. Click the **Applications** tab. This should show you the SIM and SIM-CLIENT applications.
4. Click the **Application Defined MBeans** icon for the SIM application. This will display the Application MBeans defined by SIM.
5. Click 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.

Integration Configuration Files

Integration-related configuration files might be different between Retail Integration Bus (RIB) versions (for example, RIB13.1 versus RIB13.2).

The integration-related configuration files listed in this section are for SIM integrations with other Oracle Retail products through RIB 13.2 and RSL 13.2 interfaces.

For more information, see Configuration Files.

Note: For integrating with Oracle Retail Merchandising products 13.1.x using RIB, see Appendix: SIM Integration with Oracle Retail Merchandising Products 13.1.x through RIB.

injectors.xml

This file contains mappings for RIB message injector handler class and RIB incoming message family/type.

By default, this file should not be modified.

Example 4–8 injectors.xml

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

<injector_config>
<family name="asnin">
<injector class="oracle.retail.sim_int_rib13_2.closed.SimMessageRibInjector">
<type>asnincre</type>
</injector>
<injector class="oracle.retail.sim_int_rib13_2.closed.SimMessageRibInjector">
<type>asnindel</type>
</injector>
<injector class="oracle.retail.sim_int_rib13_2.closed.SimMessageRibInjector">
<type>asninmod</type>
</injector>
</family>
….
</ injector_config>
```

jndi_providers.xml

This file contains the remote service lookup configurations to Retail Service Layer (RSL) to RMS (Retail Merchandise System) and RPM (Oracle Retail Price Management).

The configurable entries are configured by SIM installer at SIM installation. The SIM Installer also creates the security authentication wallet file and deploys to SIM application server.

Note: By default, the wallet file was created by SIM application installer and these application server user credentials are created and stored in the Oracle Wallet files in Credential Storage Manager (CSM) password store.

For more details on setting up password stores or changing user credentials in the wallet file, see the *Oracle Retail Store Inventory Management Installation Guide* for detailed information..

Example 4–9 jndi_providers.xml

```
<?xml version="1.0" ?>

<!-- This file is used for RSL only.  RIB configuration is in not included. -->

<ejb_context_overrides>
```

```
<!-- rslForRpm -->
<provider app="rpm" map-name="rpm"
                   csm-wallet="@deploy.rib-sim.wallet.dir@">
<encrypted-context-property name="java.naming.security.principal"
key="@deploy.rpm.alias@" value-type="PC_USERNAME" />
<encrypted-context-property name="java.naming.security.credentials"
key="@deploy.rpm.alias@" value-type="PC_PASSWORD" />
<context-property name="java.naming.provider.url" value="@deploy.rpm.url@" />
<context-property name="java.naming.factory.initial"
value="@deploy.rpm.factory@" />
</provider>

<!-- rslforRms -->
<provider app="rms" map-name="rms"
 csm-wallet="@deploy.rib-sim.wallet.dir@">
<encrypted-context-property name="java.naming.security.principal"
key="@deploy.rms.alias@" value-type="PC_USERNAME" />
<encrypted-context-property name="java.naming.security.credentials"
key="@deploy.rms.alias@" value-type="PC_PASSWORD" />
<context-property name="java.naming.provider.url" value="@deploy.rms.url@" />
<context-property name="java.naming.factory.initial" value="@deploy.rms.factory@"
/>
</provider>
</ejb_context_overrides>
```

The configurable values are:

@deploy.rib-sim.wallet.dir@
The location where the authentication wallet file is located on the sim server.

```
/u00/webadmin/product/10.3.3/WLS/user_projects/domains/soa_domain/sim-client/csm
```

The wallet aliases for RPM and RMS are created using the principal and credentials which must exist in the RPM Weblogic managed server and RSLForRMS Weblogic managed server respectively.

RSLForRPM:

@deploy.rpm.alias@
The wallet alias for value-types of PC_USERNAME and PC_PASSWORD.

```
rpmuser-alias
```

@deploy.rpm.url@
The RPM server's JNDI URL.

```
t3://rpm-serverxxxx:17003
```

@deploy.rpm.factory@
The InitialContextFactory class.

```
weblogic.jndi.WLInitialContextFactory
```

RSLForRMS:

@deploy.rms.alias@
The wallet alias for value-types of PC_USERNAME and PC_PASSWORD.

```
rmsuser-alias
```

@deploy.rms.url@
The RSLForRMS server's JNDI URL.

```
t3://rsl-rms-server xxxx:17007
```

@deploy.rms.factory@
The InitialContextFactory class.

```
weblogic.jndi.WLInitialContextFactory
```

> **Note:** See *Oracle Retail Store Inventory Management Installation Guide* for wallet location and configuration details.

remote_service_locator_info_ribclient.xml

This file contains the remote service lookup configuration to invoke RIB remote EJBs for publishing messages to RIB.

The SIM installer configures the configurable entries at SIM installation time. The SIM Installer also creates the security authentication wallet file and deploys to the SIM application server.

Example 4–10 remote_service_locator_info_ribclient.xml

```
<?xml version="1.0" ?>
<remote_service_locator_info>
<!-- This is for JNDI  -->
 <provider id="rib-sim" map-name="rib-sim"
    csm-wallet="@deploy.rib-sim.wallet.dir@" >
<context-property name="java.naming.factory.initial"
    value="@deploy.rib-sim.factory@"/>
<context-property name="java.naming.provider.url"
    value="@deploy.rib-sim.url@" />
<encrypted-context-property name="java.naming.security.principal"
 key="@deploy.rib-sim.alias@" value-type="PC_USERNAME" />
<encrypted-context-property name="java.naming.security.credentials"
 key="@deploy.rib-sim.alias@" value-type="PC_PASSWORD" />
</provider>
</remote_service_locator_info>
```

The configurable values are:

@deploy.rib-sim.wallet.dir@
The location where the security authentication wallet file is located on the sim server.

The wallet aliases for RIB-SIM connection are created using the principal and credentials that must exist in RIB-SIM Weblogic-managed server.

```
/u00/webadmin/product/10.3.3/WLS/user_projects/domains/soa_domain/sim-client/csm
```

@deploy.rib-sim.alias@
The wallet alias for value-types of PC_USERNAME and PC_PASSWORD.

```
ribsimweblogic-alias
```

@deploy.rib-sim.url@
The RIB-SIM server's JNDI URL.

```
t3://rib-sim-serverxxxx:19106
```
Or
```
t3://rib-sim-serverxxxx:19106/rib-sim
```

@deploy.rib-sim.factory@
The InitialContextFactory class.

```
weblogic.jndi.WLInitialContextFactory
```

> **Note:** See the *Oracle Retail Store Inventory Management Installation Guide* for wallet location and configuration details.

retail_service_config_info_ribclient.xml

This file is an RIB integration-related service factory configuration file. It specifies the RIB application's interfaces and their associated implementations.

By default, this file should not be modified.

Example 4–11 retail_service_config_info_ribclient.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<retail_service_config_info>
<!-- POJO services listing -->
<pojo-service-config>
<interface package="com.retek.rib.binding.publisher">
<impl  package="com.retek.rib.binding.publisher.impl" suffix="Impl" />
</interface>
</pojo-service-config>
<!-- EJB services listing -->
<remote-ejb-service-config>
<interface package="com.retek.rib.app.messaging.publisher.service">
<impl package="com.retek.rib.app.messaging.publisher.service.impl"
remote-bean-prefix="" remote-bean-suffix="Remote" remote-home-prefix=""
remote-home-suffix="RemoteHome" remote-service-locator-info-ref-id="rib-sim" />
 </interface>
 </remote-ejb-service-config>
</retail_service_config_info>
```

service_flavors.xml

This file is one of the Retail Platform's service factory configuration files which specifies the service request context (or flavors), for example, from a client application or the request is originated from a process executing on the server.

The services_<application>.xml file (for example, services_ribclient.xml, services_rsl.xml) specifies instructions to the Retail Platform's service factory indicating which type of the service to use based on the context of the service request.

By default, this file should not be modified

Example 4–12 service_flavors.xml

```
<?xml version="1.0" encoding="UTF-8"?>
  <services-config>
     <flavors set="client">
         <!-- Default flavor precedence and implementation -->
         <flavor name="ejb" locator="com.retek.platform.service.EjbServiceLocator"
remote-suffix="Remote" home-suffix="RemoteHome" />
         <flavor name="core"
locator="com.retek.platform.service.SimpleServiceLocator" suffix="Impl" />
         <flavor name="file"
locator="com.retek.platform.service.SimpleServiceLocator" prefix="File" />
         <flavor name="offline"
```

```
locator="com.retek.platform.service.SimpleServiceLocator" prefix="Offline" />
     </flavors>
     <flavors set="server">
         <!-- Default flavor precedence and implementation -->
         <flavor name="java"
locator="com.retek.platform.service.SimpleServiceLocator" suffix="Java" />
         <flavor name="ejblocal"
locator="com.retek.platform.service.EjbServiceLocator" remote-suffix="Local"
home-suffix="LocalHome" jndi-name-suffix="Local" />
         <flavor name="core"
locator="com.retek.platform.service.SimpleServiceLocator" suffix="Impl" />
     </flavors>
 </services-config>
```

services_rsl.xml

services_rsl.xml is the Retail Service Layer (RSL) service factory configuration file. It specifies the RSL application's services interfaces and their associated implementations. Both RSL RPM services and RSLForRMS services are included in this file.

By default, this file should not be modified.

Example 4–13 services_rsl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file contains entries in servcies_rsl.xml from RSLforRPMClient and
RSLforRMSClient -->
<services-config>
<customizations>
        <!-- rslForRpm -->
        <interface package="com.retek.platform.service" app="rpm">
            <impl package="com.retek.platform.service.impl" />
        </interface>
           <interface package="com.retek.rsl.rpm" app="rpm">
              <impl package="com.retek.rsl.rpm.impl" />
           </interface>
           <!-- rslForRms -->
<interface package="com.retek.rsl.rms.service" app="rms">
            <impl package="" />
        </interface>
        <interface package="com.retek.platform.service" app="rms">
            <impl package="com.retek.platform.service.impl" />
        </interface>
</customizations>
</services-config>
```

A

Appendix: SIM Integration Connection Troubleshooting

This appendix describes some common SIM integration connection problems, and identifies some common remedies:

RSL-to-RPM Connections

You can try the following if you are experiencing issues with the RSL-to-RPM connections:

- Verify RPM server is running.
- Verify SIM jndi_providers.xml file points to the correct RPM server.
- The username and password for the wallet alias exists in the RPM application-managed server (not applicable when integrating with RIB 13.1.x).

RSL-to-RMS Connection

You can try the following if you are experiencing issues with the RSL-to-RMS connections:

- Verify RSL-RMS server is running.
- Verify jndi_providers.xml file points to the correct RSLforRMS server.
- The username and password for the wallet alias exists in RSLforRMS application-managed server (not applicable when integrating with RIB 13.1.x).

SIM Message Publishing

You can try the following if you are experiencing issues with the SIM message publishing:

- Verify RIB-SIM server is running.
- Verify the remote_service_locator_info_ribclient.xml file in SIM server points to the correct RIB-SIM server.
- The username and password for the wallet alias used in SIM server remote_service_locator_info_ribclient.xml file exists in RIB-SIM application-managed server (not applicable when integrating with RIB 13.1.x).
- CredentialAccessPermission error: verify weblogic.policy file in SIM application server has entries for accessing sim-server cache and tmp folder (not applicable when integrating with RIB 13.1.x):

Example A–1 weblogic.policy

```
grant codeBase "file:/u00/webadmin/product/10.3.3_WLS/WLS/user_
projects/domains/soa_domain/servers/sim-server/cache/EJBCompilerCache/-" {
permission java.security.AllPermission;
permission oracle.security.jps.service.credstore.CredentialAccessPermission
"credstoressp.credstore", "read,write,update,delete";
permission oracle.security.jps.service.credstore.CredentialAccessPermission
"credstoressp.credstore.*", "read,write,update,delete";
};
grant codeBase "file:/u00/webadmin/product/10.3.3_WLS/WLS/user_
projects/domains/soa_domain/servers/sim-server/tmp/_WL_user/sim/-" {
permission java.security.AllPermission;
permission oracle.security.jps.service.credstore.CredentialAccessPermission
"credstoressp.credstore", "read,write,update,delete";
permission oracle.security.jps.service.credstore.CredentialAccessPermission
"credstoressp.credstore.*", "read,write,update,delete";
};
```

Note: The actual path can vary due to the actual location of the SIM application. The weblogic admin server needs to be bounced if weblogic.policy file is modified.

SIM Message Subscribing

You can try the following if you are experiencing issues with the SIM message subscribing:

- Verify SIM server is running
- Verify RIB-SIM server remote_service_locator_info_ribclient.xml is point to the correct SIM server
- Verify the username and password for the wallet alias used in RIB-SIM server remote_service_locator_info_ribclient.xml exists in SIM application managed server (not applicable when integrating with RIB 13.1.x).

See the Oracle Retail Integration Bus documentation for detailed troubleshooting information.

B

Appendix: SIM Integration with Oracle Retail Merchandising Products 13.1.x through RIB

This appendix describes the configuration files required to integrate SIM with supported 13.1.x Oracle Retail Products (for example, RMS 13.1.x and RPM 13.1.x, and so forth).

Note: By default, when integrating with 13.1.x Retail products (such as RMS13.1.x), the supported application server is OAS Server.

See Configuration Files for configuration file locations.

---

jndi_providers.xml

This file contains the remote service lookup configurations to Oracle Retail Service Layer (RSL) to Oracle Retail Merchandise System (RMS) and Oracle Retail Price Management (RPM).

The configurable entries are configured by SIM installer at SIM installation.

Note: By default, the wallet file was created by the SIM application installer and these application server user credentials are created and stored in the Oracle Wallet files in Credential Storage Manager (CSM) password store.

Example B–1 jndi_providers.xml

```
<?xml version="1.0" ?>

<!-- This file is used for RSL only.  RIB configuration is in jndi_providers_
ribclient.xml. -->

<ejb_context_overrides>
<provider app="rpm" url="@deploy.rpm.url@" factory="@deploy.rpm.factory@">
</provider>
<provider app="rms" url="@deploy.rms.url@" factory="@deploy.rms.factory@">
</provider>
</ejb_context_overrides>
```

The configurable values are:

RSLForRPM:

@deploy.rpm.url@
The RPM server's JNDI URL.

```
opmn:ormi://<host>:<opmn-req-port>:<rpm-oc4j-instance-name>/<rpm-app-name>
```

@deploy.rpm.factory@
The InitialContextFactory class.

```
oracle.j2ee.rmi.RMIInitialContextFactory
```

RSLForRMS:

@deploy.rms.url@
The RSLForRMS server's JNDI URL.

```
opmn:ormi://<host>:<opmn-req-port>:<rms-oc4j-instance-name>/<rms-app-name>
```

@deploy.rms.factory@
The InitialContextFactory class.

```
oracle.j2ee.rmi.RMIInitialContextFactory
```

jndi_providers_ribclient.xml

This file contains the remote service lookup configuration to invoke RIB remote EJBs for publishing messages to RIB.

The SIM installer configures the configurable entries at SIM installation time.

Example B–2 jndi_providers_ribclient.xml

```
<?xml version="1.0" ?>
<ejb_context_overrides>

    <!--
    This jndi file must be used along with services_ribclient.xml.
    -->

<provider app="rib-sim">
   <context-property name="java.naming.factory.initial"
                     value="@deploy.rib-sim.factory@" />
  <context-property name="java.naming.provider.url"
             value="@deploy.rib-sim.url@" />
  <context-property name="java.naming.security.principal"
                    value="@deploy.rib-sim.principal@" />
  <context-property name="java.naming.security.credentials"
                    value="@deploy.rib-sim.credentials@" />
    </provider>

</ejb_context_overrides>
```

The configurable values are:

@deploy.rib-sim.factory@
The InitialContextFactory class.

```
oracle.j2ee.rmi.RMIInitialContextFactory
```

@deploy.rib-sim.url@
The RIB-SIM server's JNDI URL.

```
opmn:ormi://ribsim-server:6003:rib-sim-instance/rib-sim
```

@deploy.rib-sim.principal@
The user name for connecting to rib-sim application server

```
oc4jadmin
```

@deploy.rib-sim.credentials@
The password for connecting to rib-sim application server

```
oc4jadmin
```

injectors.xml

This file contains mappings for RIB message injector handler class and RIB incoming message family/type.

By default, this file should not be modified.

Example B–3 injectors.xml

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

<injector_config>
<family name="asnin">
<injector class="oracle.retail.sim_int_rib13_2.closed.SimMessageRibInjector">
<type>asnincre</type>
</injector>
<injector class="oracle.retail.sim_int_rib13_2.closed.SimMessageRibInjector">
<type>asnindel</type>
</injector>
<injector class="oracle.retail.sim_int_rib13_2.closed.SimMessageRibInjector">
<type>asninmod</type>
</injector>
</family>
….
</ injector_config>
```

service_context_factory.xml

The service_context_factory.xml is created by sim. The RIB framework assumes that each application using the RIB has certain classes required by the Retek platform code available in the classpath.

For example, the application needs to have implementations of ServiceContextFactory and ServiceContext. These implementations do not need to actually do anything, they just need to exist. Since SIM does not use any platform code aside from that used by RIB/RSL, it does not already have these implementations available, so it needs to provide empty implementations.

By default, this file should not be modified.

Example B–4 services_context_factory.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component-config>
    <customizations >
     <interface package="com.retek.platform.service">
        <impl  package="oracle.retail.sim_int_rib13_1.closed.rib" prefix="Sim"
suffix="Impl"/>
        </interface>
```

```
    </customizations>
</component-config>
```

service_flavors_ribclient.xml

This file is one of the Retail Platform's service factory configuration files which specifies the service request context (or flavors), for example, from a client application or the request is originated from a process executing on the server.

The services_<application>.xml file (for example, services_ribclient.xml, services_rsl.xml) specifies instructions to the Retail Platform's service factory indicating which type of the service to use based on the context of the service request.

By default, this file should not be modified.

Example B–5 service_flavors_ribclient.xml

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

<services-config>
<flavors set="client">
<!-- Default flavor precedence and implementation -->
<flavor name="ejb" locator="com.retek.platform.service.EjbServiceLocator"
remote-suffix="Remote" home-suffix="RemoteHome" />
<flavor name="core" locator="com.retek.platform.service.SimpleServiceLocator"
suffix="Impl" />
</flavors>
</services-config>
```

services_ribclient.xml

This file is a RIB integration-related service factory configuration file. It specifies the RIB application's interfaces and their associated implementations.

By default, this file should not be modified.

Example B–6 services_ribclient.xml

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

<services-config>
<!--
This services_ribclient.xml file must be used along with client jndi_
providers.xml.
    -->
<customizations>
<!-- core Services -->
<interface package="com.retek.rib.binding.publisher">
   <impl  package="com.retek.rib.binding.publisher.impl" />
</interface>
<!-- app services-->
<interface package="com.retek.rib.app.messaging.publisher.service" app="rib-sim">
<impl package="com.retek.rib.app.messaging.publisher.service.impl" />
</interface>
</customizations>
</services-config>
```

services_rms.xml

The services_rms.xml in SIM is renamed from the services_rsl.xml of the RSLForRMS client jar. This file specifies the RSL application's services interfaces and their associated implementations when RSL client application communicates with RMS services (for example, Store Order, Deals, and so forth).

By default, this file should not be modified.

Example B–7 services_rms.xml

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

<services-config>
    <customizations>
     <interface package="com.retek.platform.service" app="rms">
            <impl package="com.retek.platform.service.impl" />
        </interface>

        <!-- Impl package not required here. This service is obtained    -->
        <!-- through the CommandExecutionServiceEjb which will use the   -->
        <!-- services_rsl.xml in the server side. However, the interface -->
        <!-- package is required to locate the service in the correct    -->
        <!-- application server.                                         -->
        <interface package="com.retek.rsl.rms.service" app="rms">
            <impl package="" />
        </interface>
       <interface package="com.retek.rsl.test.service" app="rms">
            <impl package="" />
        </interface>
</customizations>

</services-config>
```

services_rpm.xml

The services_rpm.xml in SIM is renamed from the services_rsl.xml of the RSLForRPM client jar. This file specifies the RSL application's services interfaces and their associated implementations for RSL client application communicates with RPM services (for example, Price Change Request, Price Inquiry, and so forth).

By default, this file should not be modified.

Example B–8 services_rpm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<services-config>
<customizations>
<interface package="com.retek.rsl.rpm" app="rpm">
<impl package="com.retek.rsl.rpm.impl" />
</interface>

<interface package="com.retek.platform.service" app="rpm">
<impl package="com.retek.platform.service.impl" />
</interface>

</customizations>
</services-config>
```

services_platform.xml

This is the Oracle Retail Platform's service factory configuration file. It specifies the mapping between the Oracle Retail Platform's core services interfaces and their associated implementations.

By default, this file should not be modified.

Example B–9 services_platform.xml

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

<services-config>
<!-- Mappings from service interfaces to implementations, -->
<customizations>
<!--interface package="com.retek.platform.service">
<impl  package="com.retek.platform.service.impl" />
</interface -->

<!-- Platform Security Services -->
<interface package="com.retek.platform.app.admin.service">
<impl  package="com.retek.platform.app.admin.service.impl" />
 …


</customizations>
</services-config>
```

services_rib.xml

This is the Oracle RIB's service factory configuration file. It specifies mapping between the RIB core services' interfaces and their associated implementations.

By default, this file should not be modified.

Example B–10 services_rib.xml

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

<services-config>

<!-- Mappings from service interfaces to implementations, -->
<!-- including any custom implementation overrides        -->
<customizations>

<!-- core Services -->
<interface package="com.retek.rib.domain.hospital.service">
<impl package="com.retek.rib.domain.hospital.service.impl" />
</interface>
<interface package="com.retek.rib.binding.injector">
   <impl package="com.retek.rib.binding.injector.impl" />
</interface>
<interface package="com.retek.rib.binding.publisher" app="rib">
   <impl package="com.retek.rib.binding.publisher.impl" />
</interface>

<!-- app services-->
<interface package="com.retek.rib.app.messaging.service">
```

```
   <impl package="com.retek.rib.app.messaging.service.impl.j2ee" />
</interface>
<interface package="com.retek.rib.app.hospital.service">
   <impl package="com.retek.rib.app.hospital.service.impl.nonj2ee" />
</interface>

</customizations>

</services-config>
```

C

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

Do the following to perform an auto-authorize unit and amount stock count:

1. Set up a product group with counting method as **Third Party** and with auto-authorize flag checked.
2. Create a new product group schedule on the Product Group screen.
3. Run the ExtractUnitAmountStockCount.sh batch program and ExtractUnitStockCount.sh batch 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. 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, including items with SOH values of zero (**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**.

4. Take a snapshot of the SOH on Stock Count List screen.

   The snapshot must be taken before uploading the third-party flat file.

   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 Third 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 Third 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**. The user 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.

5. Upload third-party count file to SIM:

   a. Once counting is complete, the third-party input file must be placed in the location specified by Oracle database directory object STOCK_COUNT_DIR.

   b. Run the ThirdPartyStockCountParser.sh batch file, passing in the name of the input file.

      See ThirdPartyStockCountParser Batch for details.

Note: The batch process 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 Rejected table. These unknown items can be assigned a valid SIM ID through the Rejected screen. 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 location specified by Oracle database directory object STOCK_COUNT_UPLOAD_DIR.

D

Appendix: Batch File Layout Specifications

This appendix describes the batch file layout specifications.

Flat File Used in the ResaFileParser Batch Process

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

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Identifies the file record type.
	File Line ID	Char(10)	0000000001	Sequential file line number.
	File type definition	Char(4)	POSU	Identifies the file type.
	File create date	Char(14)		File Create Date in YYYYMMDDHHMMSS format.
	Store	Number(10)		Store location.
	Vat include indicator	Char(1)		Determines whether or not the store stores values including vat. Not required but populated by Oracle Retail Sales Audit.
	Vat region	Number(4)		Vat region the given location is in. Not required but populated by Oracle Retail Sales Audit.
	Currency code	Char(3)		Currency of the given location. Not required but populated by Oracle Retail Sales Audit.

Record Name	Field Name	Field Type	Default Value	Description
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Not required but populated by Oracle Retail Sales Audit.
THEAD	Record descriptor	Char(5)	THEAD	Identifies the file record type.
	File Line ID	Char(10)		Sequential file line number.
	Transaction date	Char(14)		Transaction date in YYYYMMDDHHMMSS format. Corresponds to the date that the sale/return transaction was processed at Oracle Retail Point-of-Service.
	Item Type	Char(3)	REF or ITM	Can be REF or ITM.
	Item	Char(25)		ID number of the ITM or REF.
	Dept	Number(4)		Department of item sold or returned.
	Class	Number(4)		Class of item sold or returned.
	Sub Class	Number(4)		Subclass of item sold or returned.
	Pack Ind	Char(1)		Pack indicator of item sold or returned.
	Item Level	Number(1)		Item level of item sold or returned.
	Tran level	Number(1)		Transaction level of item sold or returned.
	Wastage Type	Char(6)		Wastage type of item sold or returned.
	Wastage pct	Number(12)		Waste pct (4 implied decimal places).
	Tran type	Char(1)		Transaction type code to specify whether transaction is a sale or a return.
	Drop Shipment indicator	Char(1)		Indicates whether the transaction is a drop shipment or not.
	Total sales qty	Number(12)		Total sales quantity (4 implied decimal places).
	Selling UOM	Char(4)		Selling Unit of Measure for the item.
	Sales sign	Char(1)		Determines if the Total Sales Quantity and Total Sales Value are positive or negative.

Record Name	Field Name	Field Type	Default Value	Description
	Total Sales Value	Number(20)		Total sales value of goods sold/returned (4 implied decimal places).
	Last Date time modified	Char(14)		Date and time of last modification in YYYYMMDDHHMMSS format. For VBO future use.
	Catchweight indicator	Char(1)		Indicates if item is a catchweight item.
	Total weight	Number(12)		The actual weight of the item, only populated if catchweight_ind = **Y**.
	Sub Tran type indicator	Char(1)		Tran type for ReSA. Valid values are **A**, **D**, and **NULL**.
	Total Igtax Value	Number(20)		Total Igtax Value * 10000 (four implied decimal places), goods sold or returned.
TDETL	Record descriptor	Char(5)	TDETL	Identifies the file record type.
	File Line ID	Char(10)		Sequential file line number.
	Promo Tran Type	Char(6)		Code for the promotional type from code_detail where code_type = **PRMT**.
	Promotion Number	Number(10)		Promotion number from RMS.
	Sales quantity	Number(12)		Sales quantity sold for this promotion type (4 implied decimal places).
	Sales value	Number(20)		Sales value for this promotion type (4 implied decimal places).
	Discount value	Number(20)		Discount value for this promotion type (4 implied decimal places).
	Promotion component	Number(10)		Links the promotion to additional pricing attributes.
TTAIL	Record descriptor	Char(5)	TTAIL	Identifies the file record type.
	File Line ID	Char(10)		Sequential file line number.
	Tran Record Counter	Number(6)		Number of TDETL records in this transaction set.
FTAIL	Record descriptor	Char(5)	FTAIL	Identifies the file record type.
	File Line ID	Number(10)		Sequential file line number.

Record Name	Field Name	Field Type	Default Value	Description
	File Record counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail).

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 shown in Table D–1:

Table D–1 DexnexFileParser Batch File Structure

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 (for example, 10% off) or charge (for example, 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.

Table D–2 provides details of the DexnexFileParser batch file:

Table D–2 DexnexFileParser Batch File Details

Segment	Sub-Segment	Name	Required?	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
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

Table D–2 (Cont.) DexnexFileParser Batch File Details

Segment	Sub-Segment	Name	Required?	SIM value
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

Third Party Stock Count Import File Format:

Pipe-delimited (|) file contains store count data for a store/store_count_id as shown in Table D–3:

Table D–3 Third Party Stock Count Import File

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Store Number	Char(10)		Store number file was uploaded for. It is assumed only one store is passed in per file. (Required)
	Stock Count ID	Number(12)		Unique identifier for item. Assumption is SIM will always take first stock count ID listed. (Required)

Table D–3 Third Party Stock Count Import File (Cont.)

Record Name	Field Name	Field Type	Default Value	Description
FDETL	Record Descriptor	Char(5)	FDETL	Detail record marker.
	Stock Count Date	Date(14)		Indicates date/time item was physically counted by the third party. (*YYYYMMDDHH24MISS*) For example, 20091019134600 (Required) **Note:** If not using timestamp, use **00** for time.
	Area Number	Char(10)		10-digit code indicating where in the store the item is located. (Optional)
	UPC or Item Number	Char(25)		25-digit universal product code. (Required)
	Count Quantity	Number (12,4)		Quantity counted for item, required. This field must allow for decimals when counting in UOM other than each. (Required)
	UIN(Item Serial Number)	Char(128)		Unique identification serial number for item, required if current item requires serial number.

Note: File is now delimited (users should not use pipe (|) since this will be used as a delimiter). A delimited-field file often uses less space than a fixed-length record file to store the same data. The main limitation with delimited fields is the need to add special handling to ensure that neither the field delimiter or the record delimiter characters get added into a field value.

The following is a sample Third Party Stock Count Import File:

```
FHEAD|5000|1074|
FDETL|20091014235959|1|100665085|1|ItemSerialNum1234|
FDETL|20091014140000|1|100665085|1|ItemSerialNum9999|
FDETL|20091014000000|1|100665085|1||
```

Flat File Used in Price Bulk Processing Batch Process

This includes PromotionPriceChange, ClearancePriceChange and RegularPriceChange batches. These batches import the price changes from a price management application related to PromotionPriceChange, ClearancePriceChange and RegularPriceChange in bulk.

Note: The Price Bulk Processing batch can accept an input file with a start date and end date with a time component for each promotion record. The file layout incorporates Start_date and End_date.

Start_date and End_date can accept the date and time in a **yyyymmddhh24miss** format. This batch is backward compatible with the yyyymmdd format. Only the promotion part is modified in Price Bulk Processing batch to handle time-based promotions.

For more information on the file format, see the PromotionPriceChangePublishBatch batch design, ClearancePriceChangePublishBatch batch design and RegularPriceChangePublishBatch batch design subsections in "Java and RETL Batch Processes" of the *Oracle Retail Price Management Operations Guide*.

Table D–4 ClearancePriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Line ID	Number(10)	1	Unique line ID
	File Type	Char(5)	CLRPC	Clearance Price Changes
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
FDETL	Record Descriptor	Char(5)	FDETL	File Detail Marker (one per clearance create or modify)
	Line ID	Number(10)		Unique line ID
	Event Type	Char(3)		**CRE** = Create **MOD** = Modify
	ID	Number(15)		Clearance identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		**S** = Store **W** = Warehouse
	Effective Date	Date		Clearance Effective Date (DD-MMM-YY)
	Selling Retail	Number(20,4)		Selling retail with price change applied
	Selling Retail UOM	Char(4)		Selling retail unit of measure
	Selling Retail Currency	Char(3)		Selling retail currency
	Reset Clearance ID	Number(15)		ID of clearance reset
FDELE	Record Descriptor	Char(5)	FDELE	File Detail Delete Marker (one per clearance delete)
	Line ID	Number(10)		Unique line ID
	ID	Number(15)		Clearance identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier

Table D–4 (Cont.) ClearancePriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
	Location Type	Char(1)		**S** = Store **W** = Warehouse
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line ID	Number(10)		Unique line ID
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Table D–5 RegularPriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Line ID	Number(10)	1	Unique line ID
	File Type	Char(5)	REGPC	Regular Price Changes
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
FDETL	Record Descriptor	Char(5)	FDETL	File Detail Marker (one per price change create or modify)
	Line ID	Number(10)		Unique line ID
	Event Type	Char(3)		**CRE** = Create **MOD** = Modify
	ID	Number(15)		Price Change identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		**S** = Store **W** = Warehouse
	Effective Date	Date		Effective Date of price change (DD-MMM-YY)
	Selling Unit Change Ind	Number(1)		Did selling unit retail change with this price event (**0** = no change, **1** = changed)
	Selling Retail	Number(20,4)		Selling retail with price change applied
	Selling Retail UOM	Char(4)		Selling retail unit of measure

Table D–5 (Cont.) RegularPriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
	Selling Retail Currency	Char(3)		Selling retail currency
	Multi-Unit Change Ind	Number(1)		Did multi unit retail change with this price event (**0** = no change, **1** = changed)
	Multi-Units	Number(12,4)		Number Multi Units
	Multi-Unit Retail	Number(20,4)		Multi Unit Retail
	Multi-Unit UOM	Char(4)		Multi Unit Retail Unit Of Measure
	Multi-Unit Currency	Char(3)		Multi Unit Retail Currency
FDELE	Record Descriptor	Char(5)	FDELE	File Detail Delete Marker (one per price change delete)
	Line ID	Number(10)		Unique line ID
	ID	Number(15)		Price Change identifier
	Item	Char(25)		Item identifier
	Location	Number(10)		Location identifier
	Location Type	Char(1)		**S** = Store **W**= Warehouse
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line ID	Number(10)		Unique line ID
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Table D–6 PromotionPriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record Descriptor	Char(5)	FHEAD	File head marker
	Line ID	Number(10)	1	Unique line ID
	File Type	Char(5)	PROMO	Promotions
	Export timestamp	Timestamp		System clock timestamp (YYYYMMDDHHMISS)
	Format Version	Char(5)	1.0	File Format Version
TMBPE	Record Descriptor	Char(5)	TMBPE	Promotion (transaction head)
	Line ID	Number(10)		Unique line ID

Table D–6 (Cont.) PromotionPriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
	Event Type	Char(3)		**CRE** = Create **MOD**= Modify
TPDTL	Record Descriptor	Char(5)	TPDTL	Promotion Detail Component
	Line ID	Number(10)		Unique line ID
	Promo ID	Number(10)		Promotion ID
	Promo Comp ID	Number(10)		Promotion Component ID
	Promo Name	Char(160)		Promotion Header Name
	Promo Desc	Char(640)		Promotion Header Description
	Promo Comp Desc	Char(160)		Promotion Component Name
	Promo Type	Number(2)		Promotion Component Type
	Promo Comp Detail ID	Number(10)		Promotion Component Detail ID
	Start Date	Date		Start Date & Time of Promotion Component Detail (yyyymmddhh24miss or yyyymmdd)
	End Date	Date		End Date & Time of Promotion Component Detail (yyyymmddhh24miss or yyyymmdd)
	Apply Order	Number(1)		Application Order of the Promotion
	Threshold ID	Number(6)		Threshold ID
	Customer Type ID	Number(10)		Customer Type ID
TLLST	Record Descriptor	Char(5)	TLLST	Promotion Detail Component
	Line ID	Number(10)		Unique line ID
	Location ID	Number(10)		Org Node [Store or Warehouse] identifier
	Location Type	Char(1)		Org Node Type [Store or Warehouse]
TPGRP	Record Descriptor	Char(5)	TPGRP	Promotion Detail Group
	Line ID	Number(10)		Unique line ID
	Group ID	Number(10)		Group Number
TGLIST	Record Descriptor	Char(5)	TGLIST	Promotion Group List
	Line ID	Number(10)		Unique line ID
	List ID	Number(10)		List ID
	Description	Char(120)		Description
TLITM	Record Descriptor	Char(5)	TLITM	Promotion Group List
	Line ID	Number(10)		Unique line ID
	Item ID	Char(25)		Transaction Item Identifier
TPDSC	Record Descriptor	Char(5)	TPDSC	Discount Detail for List
	Line ID	Number(10)		Unique line ID
	Change Type	Number(2)		Change Type

Table D–6 (Cont.) PromotionPriceChange Output File Layout

Record Name	Field Name	Field Type	Default Value	Description
	Change Amount	Number(20,4)		Change Amount
	Change Currency	Char(3)		Change Currency
	Change Percent	Number(20,4)		Change Percent
	Change Selling UOM	Char(4)		Change Selling UOM
	Qual Type	Number(2)		Qualification Type
	Qual Value	Number(2)		Qualification Value
	Change Duration	Number(20,4)		Change Duration
TPILSR	Record Descriptor	Char(5)	TPILSR	Items in Promotion
	Line ID	Number(10)		Unique line ID
	Item ID	Char(25)		Transaction Item Identifier
	Selling Retail	Number(20,4)		Selling retail of the item
	Selling UOM	Char(4)		Selling UOM of the item
	Location ID	Number(10)		Org Node [Store or Warehouse] identifier
TPCDT	Record Descriptor	Char(5)	TPCDT	Credit Detail
	Line ID	Number(10)		Unique line ID
	Credit Detail ID	Number(10)		Credit Detail ID
	Credit Type	Char(40)		Credit Type
	binNumberFrom	Number(10)		BinNumber From
	binNumberTo	Number(10)		Bin Number To
	Commission Rate	Number(10)		Commission Rate
	Comments	Char(160)		Comments
TTAIL	Record Descriptor	Char(5)	TTAIL	Transaction Tail
	Line ID	Number(10)		Unique line ID
FPDEL	Record Descriptor	Char(5)	FPDEL	Delete Promotion
	Line ID	Number(10)		Unique line ID
	Promo ID	Number(10)		ID of the promotion
	Promo Comp ID	Number(10)		Promotion Component ID
	Promo Comp Detail ID	Number(10)		Promotion Component Detail ID
	Group ID	Number(10)		Group Number
	List ID	Number(10)		List ID
	Item ID	Char(25)		Transaction Item Identifier for item
	Location ID	Number(10)		Org Node [Store or Warehouse] identifier
FTAIL	Record Descriptor	Char(5)	FTAIL	File tail marker
	Line ID	Number(10)		Unique line ID
	Number of lines	Number(10)		Number of lines in file not counting FHEAD and FTAIL

Table D–7 ReSA Customer Order Flat File Format

Record Name	Field Name	Field Type	Default Value	Description
FHEAD	Record descriptor	Char(5)	FHEAD	Identifies the file record type
	File Line ID	Char(10)	0000000001	Sequential file line number
	File type Definition	Char(4)	ORIN	Identifies the file type
	File Create Date	Char(14)		File Create Date in YYYYMMDDHHMMSS format
	Location	Number(10)		Store location number
THEAD	Record descriptor	Char(5)	THEAD	Identifies the file record type
	File Line ID	Char(10)		Sequential file line number
	Transaction Date & Time	Char(14)	Transaction Date	Date and time of the order processed.
	Transaction Type	Char(6)	SALE RETURN	Transaction type code specifies whether the transaction is sale or Return.
	Customer Order number	Char(30)		Customer Order number
TDETL	Record descriptor	Char(5)	TDETL	Identifies the file record type
	File Line ID	Char(10)		Sequential file line number
	Item Type	Char(3)	REF or ITM	Can be REF or ITM
	Item	Char(25)		ID number of the ITM or REF
	Item Status	Char(6)	LIN - Layaway Initiate LCA – Layaway Cancel LCO – Layaway Complete PVLCO – Post void of Layaway complete ORI – Pickup/delivery Initiate ORC - Pickup/delivery Cancel ORD - Pickup/delivery Complete PVORD – Post void of Pick-up/delivery complete S – Sale R – Return	Type of transaction.
	Dept	Number(4)		Department of item sold or returned.
	Class	Number(4)		Class of item sold or returned.
	Sub class	Number(4)		Subclass of item sold or returned.
	Pack Ind	Char(1)		Pack indicator of item sold or returned.

Table D–7 (Cont.) ReSA Customer Order Flat File Format

Record Name	Field Name	Field Type	Default Value	Description
	Quantity Sign	Char(1)	P or N	Sign of the quantity.
	Quantity		Number(12)	quantity * 10000 (4 implied decimal places), number of units for the given order (item) status.
	Selling UOM	Char(4)		UOM at which this item was sold.
	Catchweight Ind	Char(1)		Indicates if the item is a catchweight item. Valid values are **Y** or **NULL**.
TTAIL	File Type Record Descriptor	TTAILChar(5)		Identifies file record type
	File Line Identifier	Number(10)	Specified by ReSA	ID of current line being processed by input file.
	Transaction count	Number(6)	Specified by ReSA	Number of TDETL records in this transaction set.
FTAIL	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	Specified by external System	ID of current line being processed by input file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between FHEAD & FTAIL)

Note: ReSA sends in flat file to reserve and un-reserve inventory in SIM. The Customer Orders flat file will not have regular sales and return transactions.

E

Appendix: Stock Count Results Upload File Layout Specification

This appendix describes the stock count results upload file layout specification.

Stock Count Results — Flat File Specification

Once a stock count is authorized and completed, the stock count results upload file will be generated when the user authorizes a stock count from SIM PC or an auto-authorized ThirdPartyStockCountFileParser batch process.

The location of the generated output file is specified by STOCK_COUNT_UPLOAD_DIR database directory object. This database directory object and the actual directory is created when SIM is installed through SIM installer.

The directory must be on SIM database server. Once the database directory object is created, the actual file location can be found by running the following query:

```
select * from dba_directories where directory_name = 'STOCK_COUNT_UPLOAD_DIR';
```

RMS stock upload module can upload this file to update their inventory with the actual physical stock count data as shown in Table E–1.

Table E–1 Stock Count Results Flat File

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)YYYYMMDDHHMISS	date written by convert program
	stocktake_date	Date(14)YYYYMMDDHHMISSs	take_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**

Table E–1 (Cont.) Stock Count Results Flat File

Record Name	Field Name	Field Type	Description
	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
	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.

F

Appendix: Quick Print / Multiple Print

To speed up Item Ticket printing, the ticket label and printer information is maintained in a Ticket Print session. Once a user selects the label and printer in a session, the label and printer are reused for further print requests in that session. A ticket print session starts when the user chooses the Item Tickets menu option, and ends when the user exits.

When a user scans an item/UIN, all the basic validations remain intact. If all the validations pass, the Format List Screen is displayed if it is the first print request of the session. For later printing requests, the Quantity Entry screen is displayed.

If the user has Quick Print privileges, the user can select the Quick Print option.

For more information, see "Quick Print" and "Quick Print Handheld Workflow" in the *Oracle Retail Store Inventory Management Implementation Guide, Volume 3 – Mobile Store Inventory Management*.

Print Multiple Copies

Users can print multiple copies of item tickets from the SIM PC or handheld device by specifying the quantity. A Zebra printer expects a different input format for printing multiple copies compared to a regular printer. SIM provides a report parameter, COPIES, which specifies the number of copies to be printed. All SIM Item Ticket BIP reports include a common parameter (COPIES) to carry the number of copies (or quantity). The default value is 1.

The retailer needs to work with the Zebra team to implement a suitable approach to achieve multiple copy print functionality. As the final implementation is vendor-specific, the implementation cannot be certified by Oracle Retail.

G

Appendix: Item Lookup

Item Lookup allows the user to view an image of the current item by selecting the **Image** button from the Item Detail screen. If multiple images exist for the item, the user can navigate back and forth to view the different images.

Default Dao Implementation: ItemImageOracleDao

The default implementation reads the URLs for the item from the database and orders them by their display sequence. The default implementation then creates a JAVA URL object out of the value of the database and finally creates an ImageIcon using the JAVA URL object.

Customizing The ItemImageDao

To customize the logic that retrieves images, code a JAVA class that implements the ItemImageDao interface, and then alter the dao.cfg file:

Example, in the dao.cfg file:

```
ITEM_IMAGE_DAO=oracle.retail.sim.shared.dataaccess.dao.ItemImageOracleDao
```

becomes the following:

```
ITEM_IMAGE_DAO=oracle.retail.sim.shared.dataaccess.dao.CustomImageOracleDao
```

The only restriction on customizing (that is, where and how you get images) is that this customization must return a list of ImageIcon Java objects.

H

Appendix: Finishers

SIM integrates with external finishers as if they are a warehouse. This means that SIM can return items to finisher entities, or receive items from a finisher entity. These returns or receipts can be generated from external systems or internally in SIM.

In addition to the ability to ship or receive against an external finisher, SIM also subscribes to the context field on the transfer dialogue and in the returns dialogue. The context field is an additional field that is displayed to the user indicating what the purpose of the return is. This identifies which items are shipped to the external finisher when the return is created in the external system.

For example, a repair return is used to indicate to the user that items need to be sent to the external finisher for repair purposes. The external finisher can be used on the repair return to have items updated.

Business Process Flows

The following images describe the Finishers process flows.

Figure H–1 Multi-Leg Transfers

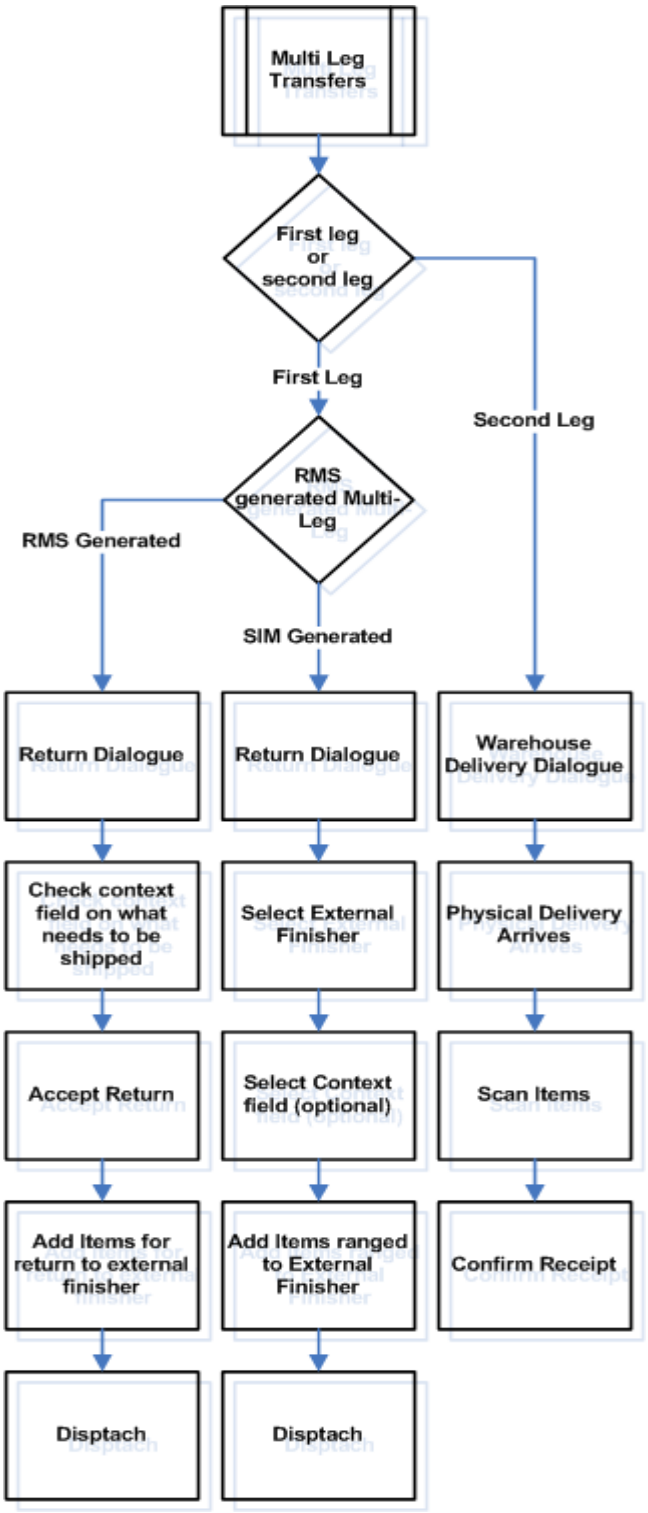


Figure H–2 SIM Originating External Finisher Transfer Dataflow Model

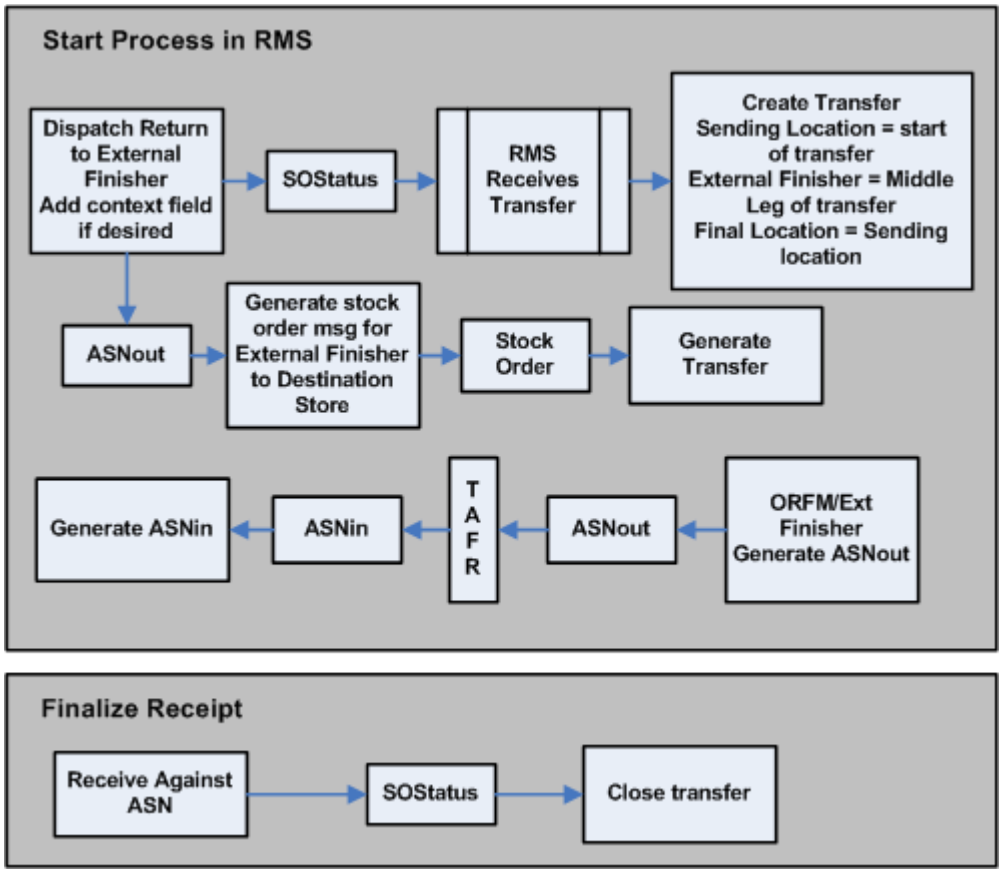
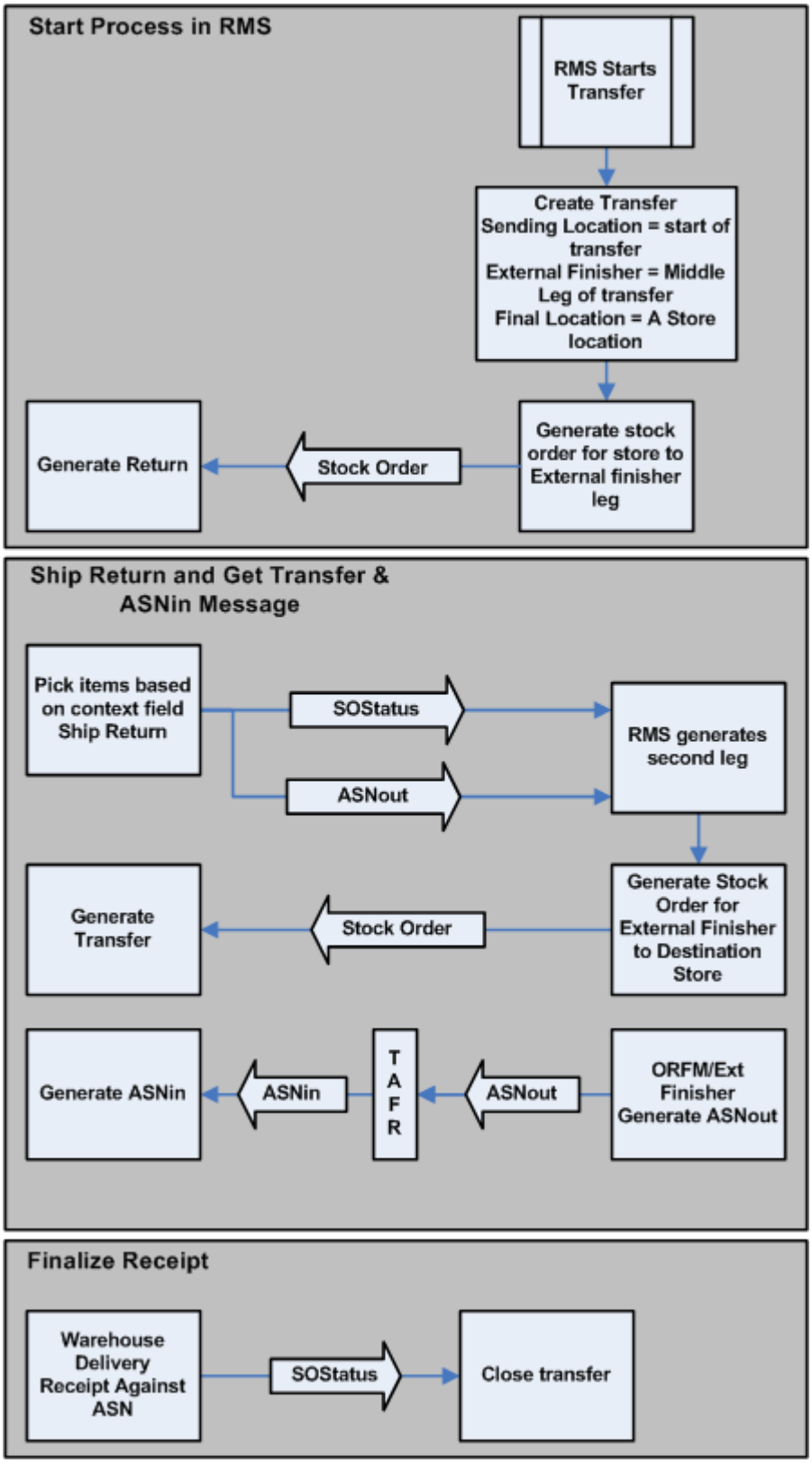


Figure H–3 RMS Originating External Finisher Transfer Dataflow Model



Index

A

B

C

D

F

I

L

P

Q

R

S

T