Oracle® Retail Store Inventory Management

Operations Guide

Release 13.1

June 2009

ORACLE®

Oracle Retail Store Inventory Management Operations Guide, Release 13.1

Primary Author: Graham Fredrickson

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

The following restrictions and provisions only apply to the programs referred to in this section and licensed to you. You acknowledge that the programs may contain third party software (VAR applications) licensed to Oracle. Depending upon your product and its version number, the VAR applications may include:

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

(ii) the **MicroStrategy** Components developed and licensed by MicroStrategy Services Corporation (MicroStrategy) of McLean, Virginia to Oracle and imbedded in the MicroStrategy for Oracle Retail Data Warehouse and MicroStrategy for Oracle Retail Planning & Optimization applications.

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

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

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

(vi) the software component known as **Access Via™** licensed by Access Via of Seattle, Washington, and imbedded in Oracle Retail Signs and Oracle Retail Labels and Tags.

(vii) the software component known as **Adobe Flex™** licensed by Adobe Systems Incorporated of San Jose, California, and imbedded in Oracle Retail Promotion Planning & Optimization application.

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

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

You acknowledge and confirm that Oracle grants you use of only the object code of the VAR Applications. Oracle will not deliver source code to the VAR Applications to you. Notwithstanding any other term or condition of the agreement and this ordering document, you shall not cause or permit alteration of any VAR Applications. For purposes of this section, "alteration" refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

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

Contents

3 Technical Architecture

4 Batch Processes

List of Figures

List of Tables

Preface

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

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

Audience

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

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

Related Documents

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

- *Oracle Retail Store Inventory Management Release Notes*
- *Oracle Retail Store Inventory Management Installation Guide*
- *Oracle Retail Store Inventory Management User Guide*
- *Oracle Retail Store Inventory Management Online Help*
- *Oracle Retail Store Inventory Management Data Model*
- *Oracle Retail Store Inventory Management Implementation Guide – Volume 1*

- *Oracle Retail Store Inventory Management Implementation Guide – Volume 2 – Integration Information*
- *Oracle Retail Store Inventory Management Implementation Guide – Volume 3 – Mobile Store Inventory Management*
- *Oracle Retail Store Inventory Licensing Information*
- Oracle Retail Service Layer documentation
- Oracle Retail Integration Bus documentation
- Oracle Retail Price Management documentation

Customer Support

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

- https://metalink.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

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

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

Conventions

The following text conventions are used in this document:

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

1
Introduction

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

Overview

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

The SIM solution 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

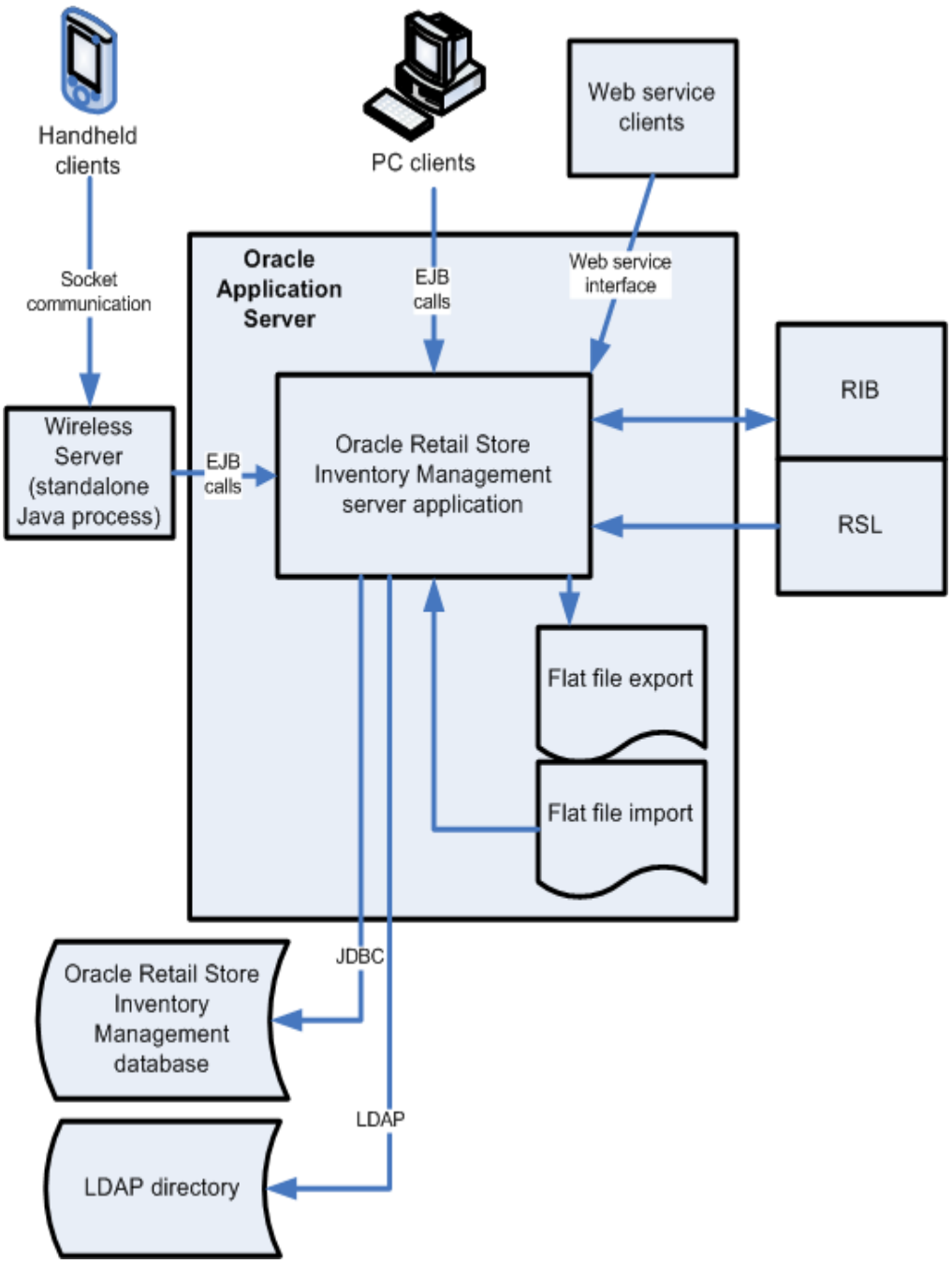
SIM's robust distributed computing platform enables enhanced performance and allows for scalability.

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 talk to the server across the wire in almost real time.

The following diagram offers a high-level conceptual view of the main components and integration points of the SIM architecture. For a detailed description of this diagram, see Chapter 3, "Technical Architecture".

Figure 1–1 SIM Technical Architecture



2

Backend System Configuration

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

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

Configuring SIM Across Time Zones

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

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

> **Note:** When integrating with Oracle Retail Price Management for pricing information, the **Enable GMT for Price Changes** option must always be set to **no** because Oracle Retail Price Management 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 above, once retailers have specified the local (store) time, they can specify which time zone, GMT or store, to use for timestamp publication to the RIB.

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

Supported Oracle Retail Products/Environments

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

For information about requirements for SIM's client, 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.

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

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

batch_db.cfg

Database connection info for batch programs.

This file is no longer used.

dao.cfg

Data access object implementations.

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

date.cfg

Date format configuration.

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

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

The prefix before the pattern key is the ISO standard two-character code for language and then two-character code for country. Both language and country must be present.

Additional language/country combinations may be added as desired (for example, frUS is French in United States).

integration.cfg

Integration (RIB and RSL) settings.

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

- ribMessagePublishEnabled – if set to **true**, SIM will actually publish messages to the RIB during processing. If set to **false**, SIM will not publish messages to the RIB, but will instead log the messages to the SIM server log file. This is intended to be used only for troubleshooting purposes. For an integrated production environment, the value should be **true**.
- rslCallsEnabled – if set to **true**, SIM will actually make RSL calls during processing. If set to **false**, SIM will not allow the user to access areas of the application that call RSL. This is intended to be used only for troubleshooting purposes. For an integrated production environment, the value should be **true**.
- *_PUB – the various keys that end in _PUB are the class names of classes that implement interfaces to publish messages to the RIB. If a retailer customizes SIM, they may need to change some of the class names in this file.
- RMS_VERSION – this key is no longer used.

jndi.cfg

JNDI settings.

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

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

  However, java processes that are clients to the SIM server (the wireless server and the java batch programs), the other keys are used to determine the JNDI information for looking up the SIM server's EJBs:
- INITIAL_CONTEXT_FACTORY – the name of the factory used to get an initial JNDI context. This should not be changed.
- OBJECT_FACTORY_PACKAGES – the java packages containing object factories. This should not be changed.
- NAMING_SERVER_URL – the JNDI URL for the naming server. This should be configured to point at the SIM server's JNDI URL. The SIM installer should have set this.
- SECURITY_PRINCIPAL – the user name to connect to the Oracle Application Server's JNDI context. The SIM installer should have set this.
- SECURITY_CREDENTIALS – the password to connect to the Oracle Application Server's JNDI context. The SIM installer should have set this.

ldap.cfg

Configuration for connecting to an LDAP server.

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

log4j.xml

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

reporting.cfg

Configuration for printing reports.

See the Reporting chapter in the *Oracle Retail Store Inventory Management Implementation Guide – Volume 1* for more information about this file.

services.cfg

Service implementation classes.

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

sim.cfg

SIM server configuration.

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

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

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

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

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

- RPM_USER_LAST_NAME – the last name of the user connecting to RPM.

Parameters Related To Server-Side Cache Refresh Rates

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

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

Parameters Related To Batch Processing Operation

- BATCH_NUM_THREADS_IN_POOL
- DEXNEX_INPUT_DIR
- DEXNEX_ERROR_DIR
- RESA_FILE_DIR
- RESA_FILE_ORIG_DIR
- RESA_TRANSACTION_SIZE

See Chapter 4, "Batch Processes" for more details about these parameters.

wireless_client_master.cfg

Wireless Server Configuration.

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

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

wireless_services.cfg

This file is no longer used.

retek/jndi_providers.xml

JNDI Configuration File:

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

retek/jndi_providers_ribclient.xml

JNDI Configuration File for RIB Communication:

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

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

retek/rules_sim.xml

Business Rules configuration.

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

retek/rib/injectors.xml

RIB subscriber configuration.

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

Port Configuration

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

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

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

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

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

- *Oracle® Application Server Administrator's Guide*
  - Section D.2 Port Numbers (Sorted by Port Number) - Shows the port ranges assigned by default.

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

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

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

Configuring the Transaction Timeout for SIM

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

- *Oracle® Containers for J2EE Services Guide*
  - Section 5.3 "Configuring the OC4J Transaction Manager"

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

Logging Information

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

The log files are configured to roll over once they reach a certain size (currently 10 MB). Once a log file reaches the configured size, it will be renamed (for example, sim.log will be renamed to sim.log.1) and new log messages will be written to a new file (for example, sim.log). If there are already rolled-over logs, they will be also be renamed for example, sim.log.1 becomes sim.log.2, sim.log.2 becomes sim.log.3, 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.

For information about logging related to the DexnexFileParser batch process, see Chapter 4, "Batch Processes".

Default Location of Log Files

Server Log Files

The log file for the server is located at:

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

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

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

`<sim-oc4j-instance>/sim-home/log/sim-batch.log`
It can be changed by changing the value of the **File** param in the `sim.appender` appender in sim-home/batch-config/log4j.xml.

Client Log Files

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

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

Changing Logging Levels

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

Editing log4j.xml

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

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

Using Oracle Enterprise Manager Application Server Control

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

Do the following to find this Mbean:

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

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

3

Technical Architecture

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

SIM Technology Stack

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

Advantages of the Architecture

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

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

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

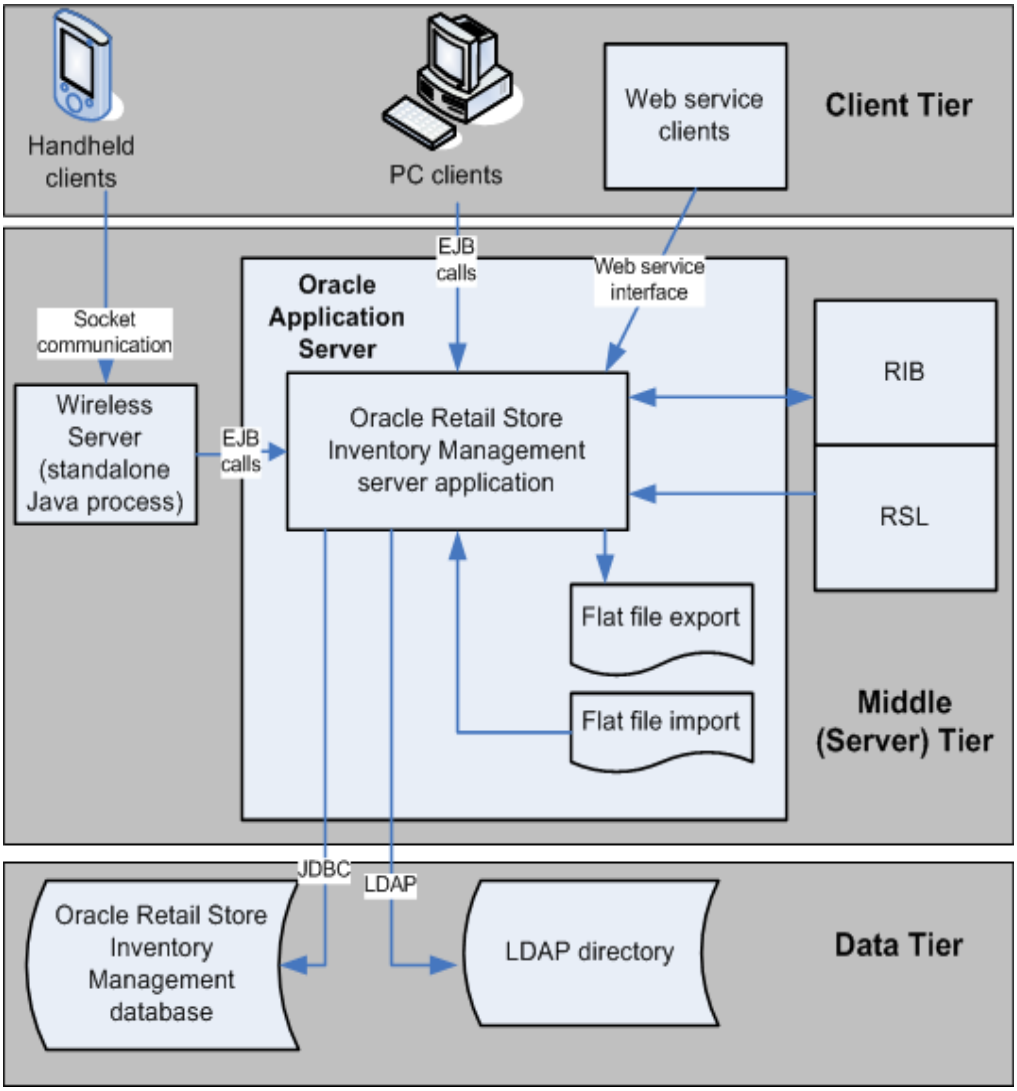
- Scalability: Hardware and software can be added to meet retailer requirements for each of the tiers.
- Maintainability: The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- Platform independence: The code is written once but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

- Endurance: Multi-tiered physically distributed architecture extends the life of the system.
- Flexibility: The system allocates resources dynamically based on the workload.

SIM Technical Architecture Diagrams and Description

This section provides a high-level overview of SIM's technical architecture. The following diagram illustrates the major pieces of the typical three-tiered SIM implementation. Descriptions follow the diagram.

Figure 3–1 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 via the industry standard Java database connectivity (JDBC) protocol. In order for the SIM client to retrieve the desired data from the database, a JDBC connection must exist between the middle tier and the database. JDBC facilitates the communication between a Java application and a relational database. In essence, JDBC is a set of application programming interfaces (API)s that offer a database-independent means of extracting and/or inserting data to or from a database. To perform those insertions and extractions, SQL code also resides in this tier facilitating create, read, update, and delete actions.

Database Tier

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

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

Distributed Topology

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

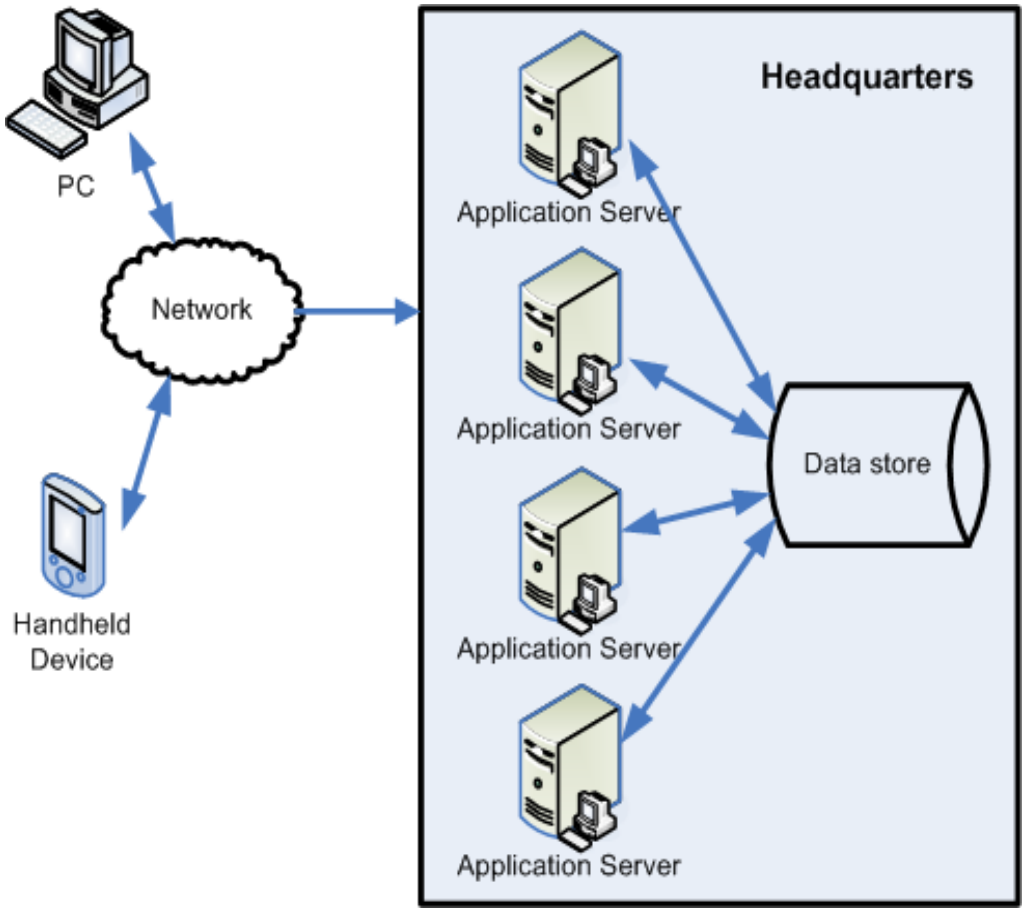
SIM's client code makes use of helper and framework classes that contain the logic to look up remote references to EJBs on the server and make calls to them. These helper and framework contain no business logic but contain only enough code to communicate with the server.

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

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

The following diagram illustrates SIM's deployment.

Figure 3–2 SIM Deployment



A Word About Activity Locking

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

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

4

Batch Processes

This chapter provides the following:

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

Batch Processing Overview

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

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

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

Running a Batch Process

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

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

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

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

Summary of Executable Shell Scripts, Batch Files, Java Packages

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

Table 4–1 **Summary of Executable Shell Scripts, Batch Files, Java Packages**

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

Scheduler and the Command Line

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

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

Return Value Batch Standards

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

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

Batch Logging

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

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.

Functional Descriptions and Dependencies

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

Table 4–2 **Batch Process Business Functionality and Dependencies**

Batch process	Description	Batch dependencies
ActivatePriceChanges	This batch process activates price changes which are effective today or on the user specified date.	No dependencies
CleanupPickList	The end of day batch process runs at the end of each day to reset the delivery bay and close any open pending pick lists.	No dependencies
ClearancePriceChange	This batch process imports the clearance price changes setup in RPM. SIM uses this data to update the price information of the items.	No dependencies
CloseProdGroupSchedule	This batch process closes the product group schedule.	No dependencies
DexnexFileParser	This batch imports the direct delivery shipment records (PO, shipment and receipt) from Dex/Nex files.	No dependencies
ExtractStockCount	The Extract Stock Count Batch program generates Unit stock counts or Unit and Amount stock counts.	No dependencies

Table 4–2 (Cont.) **Batch Process Business Functionality and Dependencies**

Batch process	Description	Batch dependencies
ItemRequest	The batch process generates item requests in pending or worksheet status for item request product group schedule which was scheduled for current date.	No dependencies
LateSalesInventoryAdjustmentPublishJob	LateSalesInventoryAdjustmentPublishJob process publishes the late sale inventory adjustments records to Retail Merchandise System (RMS) through the Retail Integration Bus (RIB).	No dependencies
ProblemLineStockCount	The problem line batch process goes through the list of items in the problem line group, determining which fall within the user specified parameters (negative SOH, negative available, and so forth). The system automatically creates a stock count from those items that do fall within the parameters.	No dependencies
PromotionPriceChange	This batch process imports the promotional price changes setup in RPM. SIM uses this data to update the price information of the items.	No dependencies
PurgeAdHocStockCount	This batch process deletes ad hoc stock counts with a status of **in progress**.	No dependencies
PurgeAll	This process deletes records from the SIM application that meet certain business criteria.	No dependencies
PurgeAudits	This batch process deletes audits.	No dependencies
PurgeDSDReceivings	This batch process deletes the Direct Store Delivery receiving.	No dependencies
PurgeInventoryAdjustments	This batch process deletes inventory adjustments.	No dependencies
PurgeItemRequests	This batch process deletes item requests.	No dependencies
PurgeItemTickets	This batch process deletes item tickets.	No dependencies
PurgeLockings	This batch process deletes lockings.	No dependencies
PurgePickList	This batch process deletes pick lists.	No dependencies
PurgePriceChanges	This batch process deletes price changes.	No dependencies
PurgePriceHistories	This batch process deletes price histories.	No dependencies
PurgeReceivedTransfers	This batch process deletes received transfers.	No dependencies
PurgeStockCounts	This batch process deletes stock counts.	No dependencies
PurgeStockReturns	This batch process deletes stock returns.	No dependencies
PurgeWHDReceivings	This batch process deletes the Warehouse delivery receivings.	No dependencies
RegularPriceChange	This batch process imports the permanent/regular price changes setup in RPM. SIM uses this data to update price information of the items.	No dependencies
ResaFileParser	This batch process imports sales and returns data that originates in a point of sale (POS) system. SIM uses the data to update the SOH for the store/items combinations in each file.	No dependencies

Table 4–2 (Cont.) **Batch Process Business Functionality and Dependencies**

Batch process	Description	Batch dependencies
ReturnNotAfterDateAlert	This batch process warns users **x** number of days in advance that the RTV/RTW is about to reach the Not After Date and must be dispatched. Note that the **x** value is configurable via the system's administration GUI screens.	No dependencies
ThirdPartyStockCountParser	This batch process imports stock count file from a third-party counting system (such as RGIS), the stock on hand quantities are updated for the existing unit and amount stock count records in SIM.	No dependencies
WastageInventoryAdjustments	This batch process looks for wastage product groups that are scheduled for today and creates an inventory adjustment for each item in the product group.	No dependencies
WastageInventoryAdjustment PublishJob	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

There are no batch dependencies in this release.

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

Batch Details

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

Activate PriceChanges Batch

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

Usage

The following command runs the ActivatePriceChanges batch job:

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

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

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

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

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

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

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.

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
```

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

ExtractStockCount Batch

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

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

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

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

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

Usage

The following command runs the ExtractStockCount batch:

`ExtractStockCount.sh <extract_date>`
Where the extract_date is optional, if specified, it must be in format of dd/mm/yyyy.

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

ItemRequest

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
```

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.

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
```

ResaFileParser Batch

This batch program imports sales transaction data (POSU file) that originated in a point-of-sale system. 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.

Batch Design Overview

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.

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.

Usage

The following pre-setup items are required when running ResaFileParser batch:

- The ReSA File Parser batch processes ReSA data files through the Oracle database stored procedure. The stored procedure locates the file location through database directory objects.
- RESA_DIR, RESA_ORIGINAL_DIR and RESA_LOG_DIR database directory objects are created when SIM is installed.
- The read and write privileges on these database directory objects must be granted to the SIM database user.
- The corresponding operating system directories for the file storage must be created.

- The ReSA data file needs to reside on the database server or at locations that can be accessed by Oracle database process. The Oracle process should have full access to the directory locations specified by RESA_DIR, RESA_ORIGINAL_DIR and RESA_LOG_DIR.
- The actual file location on the database server can be found by executing the following queries:
- Select directory_name, directory_path from dba_directories where directory_name in ('RESA_DIR', 'RESA_ORIGINAL_DIR', 'RESA_LOG_DIR');
- The system or database administrator must ensure that the operation system directory has the correct read and write permissions for the Oracle database processes.
- 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.

Use the following command to run the ResaFileParser.sh batch:

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

Where:

- *filename* (required): The name of the input POSU file containing the sales transaction data from one store.

ResaFileParser batch has the ability of processing multiple files. File names are separated by a space.

Restart and Recover

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

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

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
```

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.

Non-Auto-Authorized Count -- if the Auto-Authorize flag is not checked during product group setup, the following occurs:

- The import file contains item and quantity counted information. SIM populates the count quantity on the stock count records and sets the authorize quantity equal to the count quantity. Once the file has been imported from the RGIS system, the stock count records type is set to **authorize** and the status is set to **in progress**. The user needs to manually authorize the stock after the import process completes.
- If any items are sent from RGIS that were not already ranged to the store, SIM adds the item to the appropriate stock count record (based on department), and sets the snapshot SOH amount to 0.
- During the import process from RGIS to SIM, any unknown item data is written to the Not On File table.
- Any not-on-file or not-at-store items can be assigned a valid SIM item ID using the Rejected Items screen on the PC.

Auto-Authorized Count -- if the Auto-Authorize flag is checked during product group setup, the following occurs:

- The import file contains item and quantity counted information. SIM populates the count quantity on the stock count records, and sets the authorize quantity equal to the count quantity. Once the file has been imported from the RGIS system, the stock count records type is set to **authorize** and the status is set to **completed**.
- If any items are sent from RGIS that were not already ranged to the store, SIM adds the item to the appropriate stock count record (based on department), and sets the snapshot SOH amount to 0.
- During the import process from RGIS to SIM, any unknown item data is written to the Not On File table.

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

Usage

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

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

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

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 needs to 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 777 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.

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.

Usage

Following command runs the WastageInventoryAdjustments batch:

```
WastageInventoryAdjustments.sh
```

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

WastageInventoryAdjustmentPublishJob

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 below:

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

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

PurgeAll Batch

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

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

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

Usage

`PurgeAll.sh <purge_date>`

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

PurgeAdHocStockCount Batch

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

Usage

```
PurgeAdHocStockCount.sh
```

PurgeAudits

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

Usage

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

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

PurgeDSDreceivings Batch

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 in *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 in *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 in *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 in *dd/mm/yyyy* if purge_date is specified.

PurgeLocking 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 in *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 in *dd/mm/yyyy* if purge_date 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 in *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 in *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 in *dd/mm/yyyy* if purge_date is specified.

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 in *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 in *dd/mm/yyyy* if purge_date is specified.

PurgeWHDReceivings Batch

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

Usage

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

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

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 RPM related to Regular, Promotion and Clearance in bulk. Price Bulk Processing performs the following:

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

Running A Script

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

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

  ```
  rw_rw_r__
  ```

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

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

The following 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 RPM in BPP_INPUT_DIR directory.
2. Run the PromotionPriceChange.sh at the command prompt with price change input files as arguments separated by space. For example:

   ```
   PromotionPriceChange.sh <<filename1>> <<filename2>>
   ```

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

   Note: If no filename is passed when the batch is run, the batch processes any unprocessed records from the previous execution.

   - 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 sets the thread IDs to a logical unit of promotion information based on the number of threads configured for this promotion batch processing.

     Note: The PBP batch process supports multi-threading execution of the price change imports for faster processing. The number of parallel threads to execute the batch process can be configured. To configure, update the NUM_THREADS column in RK_RESTART_CONTROL table.

   - BulkPricePromotionJob calls the process_promotion procedure to process the price change information and update into SIM. After the completion of processing, BulkPricePromotionJob deletes the promotion price change information from the staging tables.

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 RPM 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 RPM 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.

<<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 the point-of-sale system as Customer Pickup, Layaway, Customer Order and Pending Purchase transactions.

In a point-of-sale system, 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 a point-of-sale system, 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 listed below. 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

- WastageInventoryAdjustments
- ItemRequest
- ProblemLineStockCount
- ExtractStockCount
- Price Bulk Processing Batches
- ThirdParty
- ResaFileParser

A

Appendix: 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 actual directory needs to 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.

Table A–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**
	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

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

Record Name	Field Name	Field Type	Description
	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.

B

Appendix: Batch File Layout Specifications

Flat File Used in the ResaFileParser Batch Process

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

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.
	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.

Record Name	Field Name	Field Type	Default Value	Description
	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 the point-of-sale.
	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.
	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.

Record Name	Field Name	Field Type	Default Value	Description
	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**.
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.
	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 follows:

Table B–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 (e.g. 10% off) or charge (e.g. environmental levy) information.
- LE – Contains the loop trailer.
- G84 – Contains the total quantity and cost of the delivery.
- G86 – Contains the suppliers UCC signature.
- G85 – Contains an authentication identifier.
- SE – Contains the number of transactions in the transmission.

The following table provides details of the DexnexFileParser batch file:

Table B–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

Table B–2 (Cont.) **DexnexFileParser Batch File Details**

Segment	Sub-Segment	Name	Required?	SIM value
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
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

Table B–2 (Cont.) **DexnexFileParser Batch File Details**

Segment	Sub-Segment	Name	Required?	SIM value
G84	G8402	Total Invoice Amount	Yes	Total Cost, inclusive of charges and net of allowances.
G86	G8601	Signature	Yes	Ignore
G85	G8501	Integrity Check Value	Yes	Ignore
SE	SE01	Number of Included Segments	Yes	Total # of segments between ST and SE, used for validation
SE	SE02	Transaction Set Control #	Yes	Same as ST02, used for validation
GE	GE01	Number of transaction sets included	Yes	# of sets in functional group, used for validation
GE	GE02	Group Control Number	Yes	Same as GS06, used for validation

Flat File Used in the ThirdPartyStockCountParser Batch Process

RGIS File Layout Definition

- Number of Fields: 12
- Record Length: 129

Table B–3 RGIS File Layout Definition

Data name	Field Description	Dec Length	Position from	Position to	Field type
DLSSTR	STORE NUMBER	10	1	10	Character
DLSDAT	DATE MMDDYYSSMMHH	12	11	22	Character
DLSRAN	RGIS AREA NUMBER	10	23	32	Character
DSLF12	12 CHARACTER FILLER	12	33	44	Character
DSLF13	13 CHARACTER FILLER	13	45	57	Character
DLSUPC	UPC CODE	25	58	70	Character
DLSF12	12 ZERO FILLER	12	71	82	Character
DLSQTY	COUNT QUANTITY	7	83	89	Character
DLSF01	CONSTANT OF A +	1	90	90	Character
DLSUIN	UIN FOR THE ITEM	25	91	116	Character
DLSUINTY	UIN TYPE	1	117	117	Character
DLSSTK	STOCK COUNT ID	12	118	129	Character

RGIS Sample File Data

```
121103090959591211223344555  100310831  0000000000000000222+  000000001606
0000001211030909595912112233445555  100311017  0000000000000000222+  000000001606
0000001211030909595912112233445555  100313848  0000000000000000222+  000000001606
```

Note: If ThirdPartyStockCountParser batch is running in Auto-authorize mode, a stock count results upload file will be generated.

See Appendix A, "Appendix: Stock Count Results Upload File Layout Specification" for more information.

Flat File Used in Price Bulk Processing Batch Process

This includes PromotionPriceChange, ClearancePriceChange and RegularPriceChange batches. These batches import the price changes from Oracle Retail Price Management related to PromotionPriceChange, ClearancePriceChange and RegularPriceChange in bulk.

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 B–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

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

Record Name	Field Name	Field Type	Default Value	Description
	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
	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 B–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)

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

Record Name	Field Name	Field Type	Default Value	Description
	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
	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 B–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
	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 of Promotion Component Detail (DD-MMM-YY)
	End Date	Date		End Date of Promotion Component Detail (DD-MMM-YY)
	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

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

Record Name	Field Name	Field Type	Default Value	Description
	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
	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
	Credit Detail ID	Number(10)		Credit Detail ID
	Line ID	Number(10)		Unique line 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 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

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

Record Name	Field Name	Field Type	Default Value	Description
	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 B–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

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

Record Name	Field Name	Field Type	Default Value	Description
	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.
	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.

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

Record Name	Field Name	Field Type	Default Value	Description
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.

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 ExtractStockCount.sh batch program to generate the stock counts.

   Note: After the batch has completed, from the Main Menu go to Inv Mgmt>Stock Counts>Stock Count List screen. 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 Not on File table. These unknown items can be assigned a valid SIM ID through the Not on File screen for non-auto authorized stock count. 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.

Index

A

B

C

D

E

I

J

L

P

R

S

T