Oracle® Retail Store Inventory Management

Implementation Guide, Volume 1 – Configuration, Customization and Extension

Release 13.2.3.1

December 2011

ORACLE®

Oracle Retail Store Inventory Management Implementation Guide, Volume 1 – Configuration, Customization and Extension, Release 13.2.3.1

Primary Author: Graham Fredrickson

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

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

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

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

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

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

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

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

Contents

1 Introduction

2 Technical Architecture

3 Setup and Configuration

4 Functional Design and Overviews

7 Customization

8 Internationalization

Index

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Retail Store Inventory Management Implementation Guide, Volume 1 – Configuration, Customization and Extension, Release 13.2.3

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

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

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

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

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Applications Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

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

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

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

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

Preface

The *Oracle Retail Store Inventory Management Implementation Guide, Volume 1– Overview* provides detailed information that is important when implementing SIM. The *Oracle Retail Store Inventory Management Implementation Guide, Volume 1– Overview* provides the following information and more:

- Customization instructions

  Provides details on how to extend SIM safely and correctly. Following these details mitigates the risks that SIM will cease to function when a retailer performs a customization.

- System and store administration

  Details the SIM system and store options. System option parameters allow a user to change the parameter for the entire system and all stores. Store option parameters are only specific to the store the current user is logged in to.

- Functional design and overview

  Provides detailed information concerning the various aspects of the SIM functional areas.

Audience

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

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the Oracle Retail Store Inventory Management Release 13.2.3 documentation set:

- *Oracle Retail Store Inventory Management Data Model*
- *Oracle Retail Store Inventory Management Implementation Guide, Volume 2 – Integration with Oracle Retail Applications*
- *Oracle Retail Store Inventory Management Implementation Guide, Volume 3 - Mobile Store Inventory Management*
- *Oracle Retail Store Inventory Management Installation Guide*
- *Oracle Retail Store Inventory Management Licensing Information*
- *Oracle Retail Store Inventory Management Online Help*
- *Oracle Retail Store Inventory Management Operations Guide*
- *Oracle Retail Store Inventory Management Release Notes*
- *Oracle Retail Store Inventory Management User Guide*

Customer Support

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

`https://support.oracle.com`

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 13.2) or a later patch release (for example, 13.2.2). If you are installing the base release, additional patch, and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

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

Conventions

The following text conventions are used in this document:

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

1
Introduction

Overview

Store Inventory Management (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.

Skills Needed for Implementation

The implementer needs an understanding of the applications and technical concepts described in this chapter.

Applications

Note: See the *Oracle Retail Store Inventory Management Installation Guide* for a list of the Oracle Retail applications that are certified with this version of SIM.

The implementer should understand the interface requirements of the integrated applications (with or without Retail Integration Bus) and data sources for the foundation data. Depending on the version of SIM that you are using, SIM might be deployed either as:

- Standalone (that is, without RIB)
- With RMS, RPM and RIB
- With RMS, RPM, RWMS and RIB
- With ORPOS alone
- With RMS, RPM, WMS, RIB and ORPOS

The implementer needs functional knowledge of the following applications:

- Oracle Retail Merchandising System (RMS)
- Oracle Retail Integration Bus (RIB)
- Oracle Retail Price Management (RPM)
- Oracle Retail Point-of-Service (ORPOS)
- Oracle Retail Warehouse Management System (RWMS)

Technical Concepts

The implementer should understand the following technical concepts:

- UNIX system administration, shell scripts, and job scheduling
- Web Logic application server (for Oracle Retail 13.2 deployments)
- Oracle Application Server (for integrating with RMS/RIB 13.1.x and older version only)
- Performance constraints based on the retailer's infrastructure
- Technical architecture, deployment options with load balancer
- Retailer's hierarchical (SKU/store/day) data
- Knowledge of Enterprise-Java including web services , PL/SQL
- LDAP setup and usage
- BIPublisher (Oracle printing engine) and Internet printing protocol

2

Technical Architecture

SIM Technology Stack

SIM has an n-tier architecture consisting of a client tier, a middle 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 back-end 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.

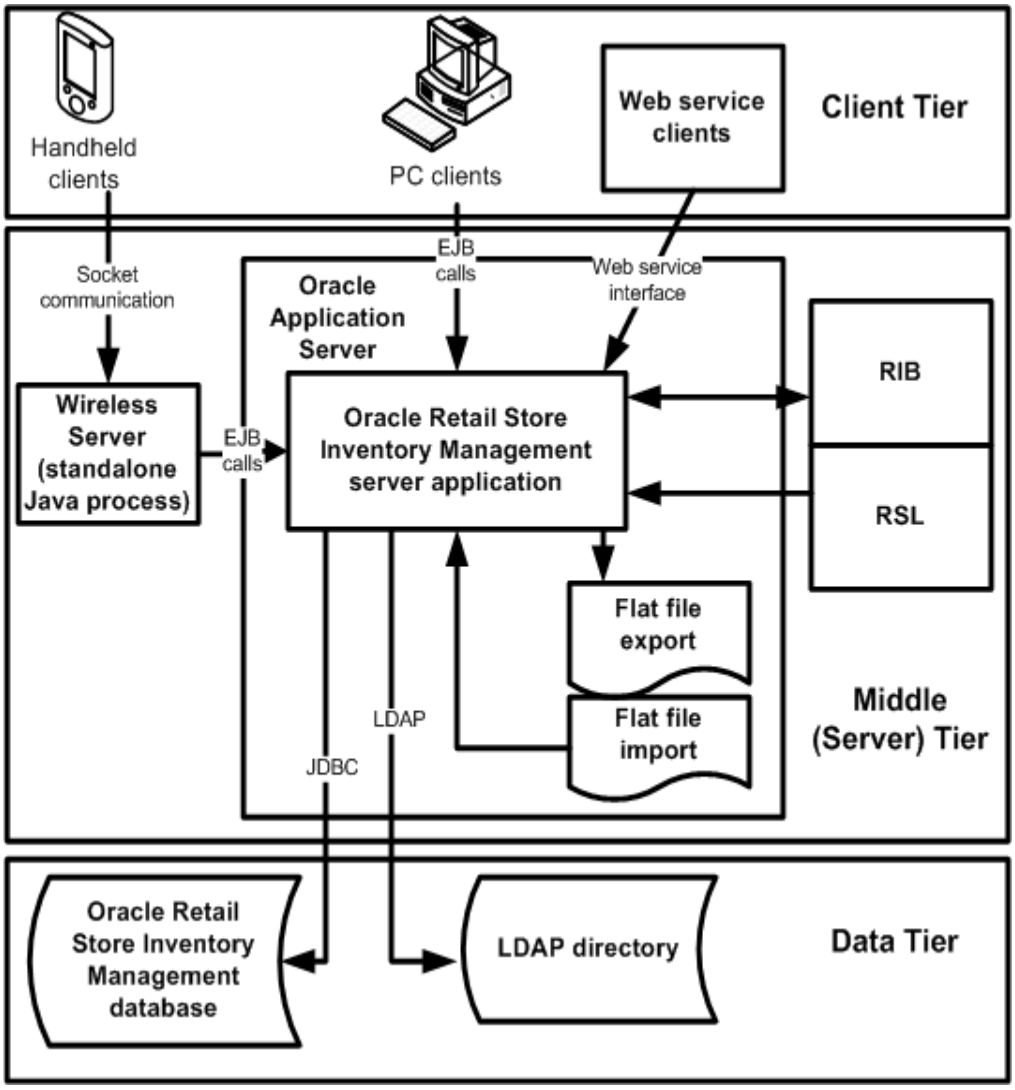
- Scalability: Hardware and software can be added to meet retailer requirements for each of the tiers.
- Maintainability: The separation of presentation, business logic, and data makes the software cleaner, more maintainable, and easier to modify.
- Platform independence: The code is written once but 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.

Figure 2–1 SIM Technical Architecture



Client Tier

SIM can be deployed on a wide variety of clients, including a desktop computer, a hand-held wireless device, and so on. The 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 graphical user interfaces (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 by themselves 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 maintain data persistence. They are used by services when database interaction is required.

Java Database Connectivity (JDBC)

DAOs communicate with the database through the industry standard Java database connectivity (JDBC) protocol. In order for the SIM client to retrieve the desired data from the database, a JDBC connection must exist between the middle tier and the database. JDBC facilitates the communication between a Java application and a relational database. In essence, JDBC is a set of application programming interfaces (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

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.

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.

Distributed Topology

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

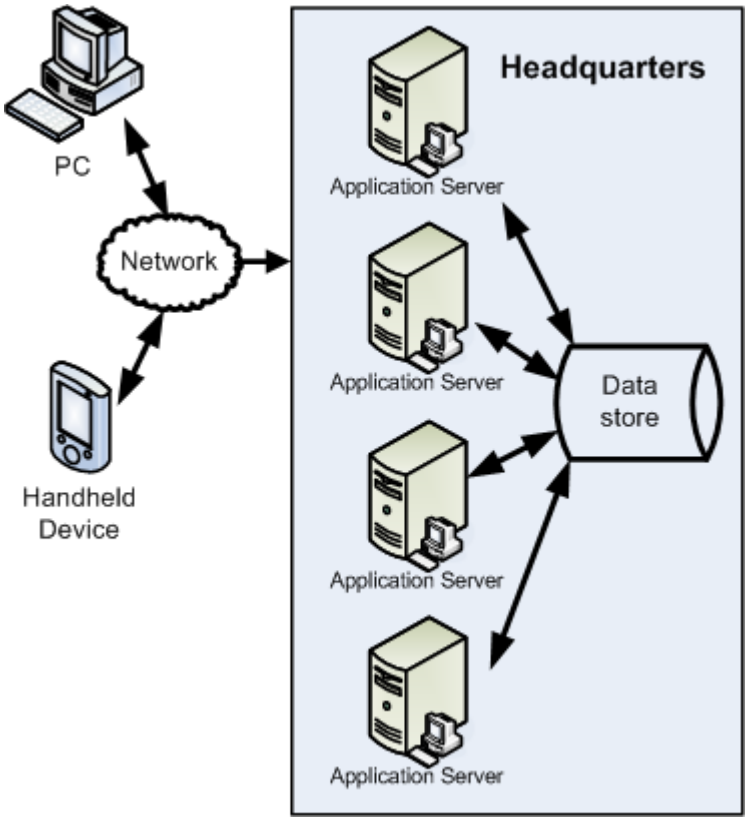
SIM's client code makes use of helper and framework classes that contain the logic to look up remote references to EJBs on the server and make calls to them. These helper and framework contain no business logic but contain only enough code to communicate with the server.

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

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

Figure 2–2, "SIM Deployment" illustrates SIM's deployment.

Figure 2–2 SIM Deployment



3

Setup and Configuration

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

Security

SIM provides role-based user access control in order to manage application functionality and data available to users.

This role-based user access control allows security to be managed in a way that corresponds closely to the organization's structure.

This model provides improved support for customization, maintenance, and management of security in SIM, simplifying customer implementations while maintaining a high degree of control and flexibility.

Security is handled by assigning privileges (permissions) to a role in SIM. These roles are assigned to stores and users (in LDAP or SIM). If the user does not have a permission the feature will not be available for user.

At this time, SIM secures buttons and drop down values on the PC and menu options on the handheld.

To allow flexibility on how security is implemented, four modes of deployment exist:

External

An external system controls security (LDAP). Users and role/store assignments are administered in LDAP. Roles are set up in SIM and need to match those set up in LDAP. Authentication is performed in LDAP.

Internal

SIM controls all aspects of security. Users, roles, user/role/store assignments are all administered in SIM. Authentication is performed in SIM.

Failover

Failover indicates that a hybrid approach will be used for authentication. Each time a user is successfully authenticated in an external system, the user information in SIM will be updated, including password. Then if the external security system is unavailable for authentication, SIM will try to authenticate the user internally with the password that was used during the last successful authentication.SIM will be able to authenticate internally created users as well in this mode.

Partial Failover

All users and roles are kept in SIM only, but the password can be handled externally. The external password can be cached in case of connection failure, but LDAP retains the master password configuration. SIM will check the password externally. If it cannot be found, SIM will look internally. If LDAP rejects the password, then the assumption is that the password is externally controlled. SIM will be able to authenticate internally created users as well in this mode.

Table 3–1 LDAP and SIM Process Control

Mode of Deployment	Application Control	Process Control
Internal		SIM: ■ User ■ Role ■ Store ■ Password ■ Login
External		LDAP: ■ User ■ Role ■ Store ■ Password ■ Login
Failover	LDAP control	LDAP: ■ User ■ Role ■ Store ■ Password ■ Login SIM: ■ Login (use cache in case LDAP is not reachable)
Failover	SIM & LDAP control	LDAP: ■ User ■ Role ■ Store ■ Password ■ Login SIM: ■ User ■ Role ■ Store ■ Login (Use Cache In Case Ldap Is Not Reachable)

Table 3–1 (Cont.) LDAP and SIM Process Control

Mode of Deployment	Application Control	Process Control
Failover	SIM Control	SIM: ■ User ■ Role ■ Store ■ Password ■ Login
Partial Failover	LDAP control	LDAP: ■ Password ■ Login SIM: ■ User ■ Role ■ Store ■ Login (Use Cache In Case Ldap Is Not Reachable)
Partial Failover	LDAP & SIM control	LDAP: ■ Password ■ Login SIM: ■ User ■ Role ■ Store ■ Login (Use Cache In Case Ldap Is Not Reachable)
Partial Failover	SIM Control	SIM: ■ User – user create ■ Role – role user assignment ■ Store – store user assignment ■ Password – password creation/maintenance ■ Login – user authentication control

How SIM Associates Menus and Menu Items

Menus and buttons on the PC are defined in the navigation.xml file. The SIM navigation framework parses and uses this xml file to display buttons. If the permission associated to the button (in the task_item_permission element) does not exist for the user, the button is not shown when the navigation menu is displayed. If the button has no associated permission then it is accessible to all users.

For more information, see Chapter 7, "Customization".

SIM Permission Definitions

The permissions used in SIM are stored in the AC_PERMISSION table. Permissions are identified by a unique name, which is used by the application to control user access and in the navigation.xml file to associate menus with permissions.

Permissions can be associated with a device type (PC, handheld, server) which is used to retrieve a user's authorized permissions during log in. When a user logs in on the PC client, only permissions with a device type of PC (or no device type) are available to the user.

Permissions can be associated with a permission group, which are stored in the AC_PERMISSION_GROUP table. Permission groups are sets of permissions that allow permissions to be filtered by category during role creation or searches.

For more information, see Appendix A, "Appendix: SIM Permissions".

SIM Role Definitions

A role is a named collection of permissions. Roles are created and edited in SIM using the security administration screens, and are stored in the AC_ROLE, AC_ROLE_PERMISSION tables. When using external security, the role header information is also stored in LDAP as a simRole, although only the roleName is used by SIM and the role information is retrieved from the SIM database. Roles can contain any combination of available permissions and can overlap with other roles.

Roles are associated with a role type, which is defined in the AC_ROLE_TYPE table. The default role types include Store and Corporate. Role types are used to control which roles a user is allowed to assign based on their permissions. A user with permission to assign store roles is not allowed to assign corporate roles without additional permissions.

The role detail screen also allows the assignment of data permissions, which control access to specific types of data. For example, data permissions can be used to control access to specific inventory adjustment reason codes, item request timeslots, role types, or product group types.

In case failover or partial failover is used, LDAP will only need to store those roles assigned to users that are controlled by LDAP.

Technical Overview

User

This class represents the header information for a user. This includes information such as:

- username
- first name
- last name
- locale
- user type
- user status
- start/end dates
- default store ID
- other state information

User objects are used to hold both internal and cached user data. Users are primarily identified by their username instead of their database ID.

UserPassword
This class represents a user password. It contains information for an individual password record such as date and status. It is used for both current passwords and password history. User password objects are used to hold both internal and cached password records.

UserRole
This class represents a role assignment for a user. It includes information such as start/end date, store ID, and other state information. User role objects are used to hold both internal and cached assignments. A role assignment with no specified store ID applies to all available stores.

UserStore
This class represents a store assignment for a user. Store assignments do not exist for super users as they have implicit access to all stores. User store objects are used to hold both internal and cached assignments.

Permission
This class represents an individual permission. It is mostly used when managing roles as it contains additional information used for assignment to roles, such as description, device type, and permission group. Permissions with no device type apply to all devices. Permissions are primarily identified by their name instead of their database ID.

PermissionGroup
Permission groups are used to categorize and filter permissions for filtering and display purposes. It is mostly used for security management operations.

PermissionSet
This class represents a set of permissions and any associated parameters. It is used to hold the set of permissions that have been assigned to a role. Permission sets are also used to hold the union of permissions for multiple roles that a user has been authorized to access. This class includes methods to test for the presence and absence of permissions in the set.

Role
This class represents the header information for a role. It contains the role name, description, role type, and whether an end date is required for assignment to a user. It is mostly used for security management operations. Roles are primarily identified by their name instead of their database ID.

RoleType
This class represents a role type that has been defined in the database. Role types are used for filtering and display purposes but are also used with data permissions to restrict access to functionality for certain types of roles. It is mostly used for security management operations.

External Security Setup (LDAP)

The external security model uses LDAP. In this mode LDAP is the only responsible application for all security control (with exception of assigning permissions to roles). LDAP will need to be set up before users can login.

SIM User Definitions

Users are defined in LDAP as simUser records.

User records contain information such as:

- user name
- status
- user type
- default store
- locale
- other data defined by the schema

To log in to SIM, a user must have an active status (**0**). Users can be assigned start or end dates to restrict their authorization by date.

SIM User Allowed Stores

Users are assigned stores to which they are allowed access. To log in to a store, the user must first be assigned to that store. The user's allowed stores also restrict which stores the user can be assigned roles for.

Users that are defined as super users are allowed access to all stores, but still require role assignments in order to gain permissions.

Store assignments are stored in LDAP as userStores attributes in simUser records.

When a user logs into SIM using the PC client, their default store is automatically selected. The user can change stores by selecting one of their allowed stores from the combo box on the main screen.

SIM User Role Assignments

Users are given permissions by assigning roles to users. Permissions are never directly assigned to users. A user can be assigned multiple roles, producing a combined permission set that is the union of the role permissions.

Role assignments are stored in LDAP for an external model as simUserRole records, which are child nodes of simUser records. Role assignments can have start or end dates to restrict their validity by date. The userRoleStores attribute of the simUserRole record specifies which stores are valid for the role assignment. If no store is specified then the role assignment applies to all stores available to the user.

When a user logs into SIM they are given permissions for all valid role assignments for the store that was selected.

Oracle Software Security Assurance (OSSA)

Sensitive information such as user crendentials must be encrypted and stored in a secure location, known as password stores or wallets. These password stores are secure software containers that store the encrepyted user credentials.

SIM has implmented using wallet alias names in the following areas:

- RIB remote service lookup (publisher and subscriber)
- RSL remote service lookup

SIM uses external secure password stores for the SIM client to look up SIM remote services:

- SIM stores the database password in a secure password store for the database standalone program which invoke sqlplus or sqlldr.
- SIM stores the application remote login password in a secure password store for java application programs.

SIM also modifies programs to use security alias names for accessing database or remote applications:

- The application server user name and password are listed in the jndi.cfg files
- The application server user name and password are listed in JnlpLaunch.properties file
- The data seeding (import) program passes the user and password when involking the sqlplus and sql loader (sqlldr) inside the program
- Any other data import utility or adhoc batch program uses SIM standard java wrapper to call the stored procedure; if using java wrapper is not applicable, and if connecting to the database through a database client utility such as sqlplus or sqlldr, then the secure pass store is used and tnsalias for database connection credential stored in the wallet must be used.

For more information, see "Appendix: Setting Up Password Stores with Oracle Wallet" in *Oracle Retail Store Inventory Management Installation Guide*.

Setting up LDAP Data for SIM

SIM is intended to work with any Lightweight Directory Access Protocol (LDAP) product. Out of the box, SIM ships sample .ldif files that can be used to create data in an LDAP system. We expect customers to use these files as examples to create their own data load files and hook into their own pre-existing corporate LDAP authentication system.

Once an LDAP server has been installed, the SIM data schema (SIM.schema) must be loaded on top of the default LDAP core schema (core.schema) supplied by the server. The following sample LDIF files are included in this release at SIM_INSTALL_DIR/sim/application/sim13/ldap. For more information, see Appendix B, "Appendix: LDAP Schema".

> **Note:** The following scripts and configuration files are provided as examples only. Variations will be necessary to match the data setup in SIM and the LDAP server that is chosen and installed.

readme.txt
Descriptions of the files in the directory and an overview of how the data needs to be structured in LDAP.

sim_objectclasses.ldif
The objectclasses that are used and required by SIM. This file can be used directly to create the required objectclasses in your LDAP directory.

sim_add_company.ldif
The base company container. This file must be modified before it is imported into your LDAP system.

sim_add_containers.ldif
The containers for holding users, stores, and roles. This file must be modified before it is imported into your LDAP system.

sim_data_roles.ldif
Sample role data. This file must be modified before it is imported into your LDAP system.

sim_data_stores.ldif
Sample store data. This file must be modified before it is imported into your LDAP system.

sim_data_users.ldif
Sample user data. This file must be modified before it is imported into your LDAP system.

sim_data_users_roles.ldif
Sample user role assignment data. This file must be modified before it is imported into your LDAP system.

Note: A simUser can have more than one simStore by simply repeating the userStores line, but should only have one defaultStore. A simUserRole can also have more than one simStore by repeating the userRoleStores line.

Note: Any user store entry for the user object must have corresponding Store data populated in the SIM Oracle database to allow a successful login (table PA_STR_RTL). Any simUserRole object must have corresponding role and store data populated in the SIM Oracle database.

Using Oracle Virtual Directory to Authenticate SIM

This document explains how to use the Oracle Virtual Directory (OVD) to authenticate Oracle Retail Store Inventory Management.

The following document is available through My Oracle Support (formerly MetaLink).

Access My Oracle Support at the following URL:

https://support.oracle.com

Oracle Retail Store Inventory Management: Using Oracle Virtual Directory to Authenticate Oracle Retail Store Inventory Management (**Doc ID: 840179.1**)

Internal Security Setup (SIM)

The default security model in SIM is LDAP authentication (external authentication). To change the security model to use internal security, run the following SQL script:

```
update rk_config set CONFIG_VALUE = '0' where config_key = 'SECURITY_
AUTHENTICATION_METHOD';
```

For internal security, use the following:

- username: **orsimadmin**
- password: **orsimadmin**

SIM User Definitions

Users are defined in SIM through the UI.

User records contain information such as:

- user name
- status
- user type
- default store
- locale
- other data defined by the UI

To log in to SIM, a user must have an active status. Users can be assigned start or end dates to restrict their authorization by date.

SIM User Allowed Stores

Users are assigned stores to which they are allowed access. To log in to a store, the user must first be assigned to that store. The user's allowed stores also restrict which stores the user can be assigned roles for.

Users that are defined as super users are allowed access to all stores, but still require role assignments in order to gain permissions. New stores are automatically assigned to this user, but role assignments are not.

When a user logs into SIM using the PC client their default store is automatically selected. The user can change stores by selecting one of their allowed stores from the combo box on the main screen.

SIM User Role Assignments

Users are given permissions by assigning roles to users in the SIM UI. Permissions are never directly assigned to users. A user can be assigned multiple roles, producing a combined permission set that is the union of the role permissions.

Role assignments can have start or end dates to restrict their validity by date.

Since users can have different roles at different stores (for example, a manager in Store One, but sales associate in Store Two), roles and stores are assigned as a pair to a user. This allows for very specific setup in SIM.

When a user logs into SIM they are given permissions for all valid role assignments for the store that was selected.

Failover Setup (SIM/LDAP)

Failover setup will cache external user role and store assignment as well as password information. This allows users to continue to log in when LDAP is down. SIM Internal assigned stores/roles will be added to the external user assigned roles/stores. In case the user is fully internal to SIM, SIM password information will be used to authenticate.

It is optional to create a user in SIM, or assign roles and stores in SIM.

SIM User Definitions

Users are defined in SIM through the UI or in LDAP. If a user at some point will have SIM-assigned roles/stores, and corporate-assigned roles/stores, the user needs to be first created in SIM, before that user has logged in to LDAP. Once cached information is pulled down into SIM, the user can no longer be created in SIM. The password initially assigned to the user will be trumped by the password assigned in LDAP.

Users can be created externally, internally or exist both in SIM and LDAP.

SIM User Allowed Stores

Can be assigned in SIM or in LDAP, or both.

SIM User Role Assignments

Roles are assigned to SIM or LDAP, or both.

Partial Failover Setup (SIM/LDAP)

Partial failover setup will only cache password information. Users, stores and role assignment information are all handled inside of SIM.

This allows corporate control on which users can log in, but detailed authorization assignments can be controlled by a different group of users (for example, Store manager). The user will also continue to be able to log in when LDAP is down.

In case the user is fully internal to SIM, SIM password information will be used to authenticate.

SIM User Definitions

Users are defined in SIM through the user interface and in LDAP. The user has to be first created in SIM, before they should log in. The password initially assigned to the user will be replaced by the password assigned in LDAP.

Users have to exist both in LDAP and SIM, since LDAP authenticates and SIM authorizes.

SIM User Allowed Stores

Should only be assigned in SIM.

SIM User Role Assignments

Roles are assigned in SIM only.

Time Zones

When stores are added to SIM, they are populated in the PA_STR_RTL table. The store time zone column, RK_TIMEZONE, is NULL by default. When it is null, SIM assumes a GMT time zone for that store.

After SIM is initially set up with DataSeeding, all stores in PA_STR_RTL need to be updated with a valid time zone. In addition, every time a new store is added to the system (for example, from an external system through an Oracle Retail Integration Bus (RIB) message), the new record in PA_STR_RTL also needs to be updated with a valid time zone as specified in steps 1-2, following.

Setting the Time Zone (standalone)

To set the proper time zone for a store, you will need to add an appropriate value to the RK_TIMEZONE column of the PA_STR_RTL table. Since there is no GUI available for this, you will need a user with DBA privileges to make the table data modification.

1. Valid time zones can be found in the SIM view TIME_ZONE_NAMES_V.
2. Choose the appropriate value from the view, and populate that value in the RK_TIMEZONE column of the PA_STR_RTL table.
3. Repeat these steps for each store.

Examples of some time zones are:

- Asia/Tokyo
- Europe/Belfast
- GMT
- Mexico/BajaSur
- Pacific/Honolulu
- US/Central
- US/Eastern

Defaulting Store Configuration Parameters

There are a number of store options related to reporting and printing reports in SIM. These can be configured at the store level; however it is best to have reasonable default values for these options so that when new stores are created in SIM (either through data seeding or by getting a message from the RIB) the configuration is mostly correct.

The default options are created by a PL/SQL procedure called INSERT_DEFAULT_ST_CONFIG_VAL. The definition of this PL/SQL procedure should be modified before running data seeding so that it creates default options specific to your environment.

Note: This procedure can be altered to modify the default value for any store configuration option. The following are the specific keys related to reporting. For definitions of what these keys mean, see Chapter 5, "System and Store Administration".

- REPORTING_TOOL_ADDRESS
- REPORTING_TOOL_REQUEST_URL
- REPORTING_TOOL_REQUEST_USERNAME
- REPORTING_TOOL_REQUEST_PASSWORD

Data Seeding

SIM requires merchandising data such as item, supplier, stores, warehouse, hierarchy, UOM conversion, differentiators, organization unit, pricing, and configuration information for transaction data. This merchandising data needs to be imported from a merchandising system, for example, Retail Merchandising System (RMS). The SIM data seeding process seeds data into SIM through an export and import mechanism.

The seed data includes foundation data (non-store specific data, for example, item, supplier, merchandise hierarchy, and so forth) and store data (for example, store, item locale, and so forth). The data seeding provides a feature to import foundation data and all stores data; it also provides a feature to seed store-related data only for the required stores.

When completed, a verification process verifies the data seeding. The verification process counts the number of records in RMS and number of records in SIM for the same entity. Any discrepancies will be known and suitable action can be taken by the administrators based on the log.

Retailers are required to be aware of the following before starting the data seeding process:

- It is highly recommended to back up the SIM database before executing the data seeding scripts.
- Data seeding scripts flush out the data from few tables before seeding. The tables include Item, Item-Location, Supplier, Party and Party Details. This is done to have a clear set of data and avoid any conflict with the data that would exist in the tables before.
- Performance can be an issue when a large amount of data needs to be migrated in a limited timeframe.
- If there are any custom modifications to the SIM database schema, then those need to be handled by the custom-modification of the seeding scripts or retailer team.
- Any bad data, due to null values or missing constraints, identified during seeding must be manually moved to the new SIM schema.
- Verify the availability of required configuration and access to Oracle database utilities.
- For the execution of data seeding scripts, the minimal required configuration should be available.
- The data seeding scripts need access to the Oracle DB utilities such as SQL Plus and SQL loader. The access to these utilities by the control file needs to be verified. If there is no access then an error is shown to provide necessary access.
- Verify RMS and SIM Corporate Database access. Verify that both RMS and SIM databases are up. Data seeding will fail to complete if either of the databases is down.
- Verify file and folder permissions. During the execution of data seeding, there is a need to create temporary files and folders. The recommended permissions for data seeding directories or files are rwxrwxr-x(775).

Executing Script

See "Running Data Seeding" in the *Oracle Retail Store Inventory Management Installation Guide* for data seeding execution steps.

Data Seeding Scripts and Usage Description

Following are the main script files and their usage description:

Table 3–2 Script Files and Usage Description

Script Files	Description
data_seeding.sh	This is the main control script for the data seeding process. It gives the option of selecting foundation data seeding, store data seeding, data seeding foundation verification, data seeding store verification and data seeding cleanup. The main tasks of this script are: ■ Checking to see if ORACLE_HOME is set. ■ Checking to see if SQLPLUS and SQLLDR are accessible. ■ Checking to see if the SIM and RMS databases are up and accessible. ■ Getting the user input for seeding. ■ Invoking data_seed_foundation.sh if foundation data seeding is selected. ■ Invoking data_seed_store.sh if store data seeding is selected. ■ Invoking data_seed_foundation_verification.sh if data seed foundation verification is selected. ■ Invoking data_seed_store_verifiation.sh if data seed store verification is selected. ■ Invoking data_seed_cleanup.sh if data seeding cleanup is selected.
data_seed_foundation.sh	This is the main control script for foundation data seeding which invokes separate scripts and completes the data seeding for foundation data from RMS to SIM. The main tasks of this script are: ■ Initial cleanup of data (temp tables, stored procedures, functions and indexes created as a part of data seeding) by invoking data_seed_cleanup.sh. ■ Invoking initial_setup_rms.sql, initial-setup_sim.sql to carry out the initial setup of temp tables, procedures, functions and indexes. ■ Disabling the related table foreign keys by invoking disable_fks.sql. ■ Exporting the data from RMS by executing the .sql files in the export_data folder. ■ Importing the data into SIM using the control file in the control_file folder. ■ Loading the data into SIM by invoking config_to_sim.sql, item.sql, supplier.sql, warehouse.sql. ■ Enabling the foreign keys that were disabled at the start of foundation data seeding by invoking enable_fks.sql. It gives the count of tables and can be verified with the disabled FK table count. ■ Carrying out the data seeding verification by invoking data_seed_foundation_verification.sh.

Table 3–2 (Cont.) Script Files and Usage Description

Script Files	Description
data_seed_store.sh	This is the main control script for store data seeding which invokes separate scripts and completes the data seeding for store data from RMS to SIM. The main tasks of this script are: ■ Setting the default option for store selection as **ALL** (data seeding is done for all the stores). ■ Letting the user input store IDs for data seeding. The store IDs should be separated by comma, with no spaces between the store numbers entered. ■ Initial cleanup of data (temp tables, stored procedures, functions and indexes created as a part of data seeding) by invoking data_seed_cleanup.sh. ■ Invoking initial_setup_rms.sql and initial-setup_sim.sql to carry out the initial setup of temp tables, procedures, functions and indexes. For the verification process, it also invokes verification_setup_store_sim.sql to setup the tables/procedures for verification at the end of seeding. ■ Disabling the related table foreign keys by invoking disable_fks.sql. ■ Preparing store list filter for the data export from RMS. For each of the stores to be seeded, the following is done: – Creating a store_number_temp table in RMS. This temp table holds only one store ID at any given time. Stores_list.sql will call the procedure SPLIT_AND_INSERT_STORE_NUMBERS and creates a directory with the store ID as its name, in the DATA folder under STORE. – Creating a store_number_temp_verify table in RMS. This temp table holds all the store IDs that are given as input for the store data seeding. This table is used for the verification process at the end of the seeding. Stores_verification.sql will call the procedure STORE_NUMBERS_VERIFY. – Creating a store_number_temp table in SIM. This temp table holds the list of all store IDs that are given as input for the store data seeding. This table is again used for the verification process. Stores_list.sql will call the procedure SPLIT_AND_INSERT_STORE_NUMBERS to add the store ID entries to the temp table.

Table 3–2 (Cont.) Script Files and Usage Description

Script Files	Description
	■ Exporting the data from RMS by executing the .sql files in the export_data folder. ■ Importing the data into SIM using the control file in the control_file folder. ■ Loading the data into SIM by invoking stores.sql. ■ Loading the dependencies for all the seeded stores by invoking inventory_adjustment_for_unavailable_qty.sql,msob.sql, report_format.sql, sequencing.sql,store_config.sql. ■ Enabling the foreign keys that were disabled at the start of foundation data seeding by invoking enable_fks.sql. It gives the count of tables and can be verified with the disabled FK table count. ■ Carrying out the data seeding verification by invoking store_data_verification.sql, which will insert the RMS and SIM condition to the verify table and execute the procedure DATASEEDVERIFY.
data_seed_foundation_verification.sh	This is the foundation data verification script. The main tasks of this script are: ■ Invoking verification_setup_foundation.sql by passing the RMS DB connection details as parameters (RMS_USER/RMS_PWD@RMS_DB). ■ Invoking foundation_data_verification.sql.
data_seed_store_verification.sh	This is the store data verification script. The main tasks of this script are: ■ Invoking verification_setup_store.sql by passing the RMS DB connection details as parameters (RMS_USER/RMS_PWD@RMS_DB). ■ Invoking call verification_setup_store_rms.sql. ■ After the initial setup is done, the user is prompted to enter the store IDs for verification. Store IDs must be separated by a comma; no space between the store IDs. ■ Invoking store_data_verification.sql.
data_seed_cleanup.sh	This is the cleanup script. It does the entire cleanup of any temporary tables, indexes, stored procedures and functions created both on the RMS and SIM database as a part of data seeding. This script invokes data_cleanup_rms.sql and data_cleanup_sim.sql.
data_seed_uin_attributes.sh	This script inserts/updates the UIN attributes on department/class level for each store which uses UINs (indicated by the store configuration AUTO_DEFAULT_UIN_ATTRIBUTES value). The input data file created by the user is processed to set up the store UIN at department and class, as well as store UIN items. See Appendix G, "Appendix: Data Seed UIN Attributes File Layout" for details.
data_cleanup_rms.sql data_cleanup_sim.sql	Drops the temporary tables, stored procedures, functions and indexes created as a part of the data seeding process in RMS and SIM, respectively

Table 3–2 (Cont.) Script Files and Usage Description

Script Files	Description
verification_setup_foundation.sql verification_setup_store.sql	Drops any existing database link, drops temporary tables, creates a database link to RMS and creates the temporary verification table for foundation and store, respectively.
foundation_data_verification.sql store_data_verification.sql	Consists of scripts to insert the details of the tables to be verified into the temporary verification table. It calls the DATASEEDVERIFY procedure to verify the count of rows in RMS and SIM for foundation and store, respectively.
verification_setup_store_rms.sql verification_setup_store_sim.sql	Run as a part of the store data seeding to setup the temporary tables and procedures to capture the store IDs that are given as input for the store data seeding for RMS and SIM, respectively.

Note: On Solaris platforms, some standard UNIX commands exhibit abnormal behavior. If data seeding verification indicates errors, a likely cause is that the import into SIM started before earlier extract processes were completed.

This can be fixed by adding a sleep command before starting the successor process. The following is an example of adding additional sleep time in data_seed_foundation.sh:

```
# checking if RMS SQL process done
<< script >>
echo 'Importing data into SIM'

#add additional sleep time before start sucessor proccess
Sleep 900
echo 'Import data to SIM completed'

echo ctl loading data into SIM to complete `date`
<< script >>

#add additional sleep time before start sucessor proccess
Sleep 900
<< script >>

echo 'Data Loaded'
#add additional sleep time before start sucessor proccess
Sleep 900
echo 'Enabling Foreign Keys'
<< script >>
```

Performance Improvement Tips

The following tips are only suggestions, and retailers need to use them at their own discretion:

- Usage of DIRECT=TRUE will improve the performance as the SQL loader loads the data directly into tables rather than generating insert queries.

Note: DIRECT=TRUE can be used only if there are no clustered tables in the schema.

- Use Unrecoverable as an option in SQL loader that would avoid the writing of the changes to redo log files and in turn improve the performance.
- Set the destination database to No Archive Log mode.

Security FAQ

- The default security model in SIM is LDAP authentication (external authentication). To change the security model to use internal security, run the following SQL script:

  ```
  update rk_config set CONFIG_VALUE = '0' where config_key = 'SECURITY_
  AUTHENTICATION_METHOD';
  ```

  For internal security, use the following:

  – username: **orsimadmin**

  – password: **orsimadmin**

- The default algorithm used to store passwords in SIM is Secure Hashing Algorithm (SHA).

  This can be configured in the server.cfg file to be any algorithm recognized by the Java encryption API.

- When creating roles, job functions and corporate hierarchies must be considered and taken into account.

4

Functional Design and Overviews

This chapter provides information concerning the various aspects of SIM's functional areas.

Store Inventory Management 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
- Fewer markdowns

Store Inventory Management ensures that all available salespeople are on the sales floor selling to customers.

The benefits of the Store Inventory Management solution include:

- Improve customer service and coverage
- Ability to perform back office functionality anywhere in the store, even on Oracle Retail Point-of-Service terminals
- Improve perpetual inventory levels by enabling floor-based inventory management through handheld devices and store PCs
- Minimize the time required to process a receipt and check-in of incoming merchandise
- Receive, track, and transfer merchandise accurately, efficiently, and easily
- Reduce technology costs by centralizing hardware requirements
- Easy to use GUI interface guiding users through the required transactions
- Extensible technology platform that allows customizations to the product. This ensures the retailer's modifications are isolated during product upgrades and lowering the total cost of ownership.

Store Inventory Management has been specifically designed to meet the needs of a high turnover labor force by providing easy to use screens that guide a user through processing a transaction.

Store Inventory Management also provides Store managers and personnel with the ability to easily perform an array of in store operations:

- Receive merchandise
- Replenish stock
- Manage physical inventories
- Look up product information
- Transfer or return stock
- Adjust inventory
- Stock counts
- Order stock

Store Inventory Management provides store employees with the information and flexible capabilities that are needed to maintain optimal inventory levels in the store and convert shoppers into buyers.

Solution and Business Process Overview

Store Inventory Management manages the inventory movement of merchandise within the store and provides users with detailed Item/SKU information needed to perform key tasks. The functionality in SIM includes:

- Lookups – Item, Supplier, Container, Customer Order Pickup
- Unique Identification Number (UIN) Support
- Receiving – Warehouse, Supplier
- Transfers/Transfer Requests
- Returns/Return Requests
- Receiver Unit Adjustments
- Stock Counts
- Store Ordering
- Item Requests
- Sequencing
- Shelf Replenishment
- Inventory Adjustments
- Wastage
- Price Changes
- Ticketing
- Item Basket
- E-mail Alerts
- Printing Reports

SIM also has System Administration functionality, which enables users to configure different parameters within the system based on their business processes. The system administration screens also contain the ability to create product groups. A product group is a collection of departments, classes, subclasses, or items, which can be used to schedule stock counts, order product, replenish store shelves, and for addressing wastage.

SIM is fully integrated with the Oracle Retail Merchandising System (RMS), a warehouse management system, an invoice matching system, and Oracle Retail Sales Audit (ReSA) using the Oracle Retail Integration Bus (RIB). Most transactions are near real-time, with only a few using batch processes and some using real time integration. Any store using SIM, maintains its own inventory and reports those numbers to the merchandising system. Most foundation data within SIM can be populated using the data-seeding program provided.

Deploying SIM as part of the Oracle Retail enterprise ensures the accuracy and timeliness of all inventory information across the retailer's supply chain.

The SIM UI is split up in five parts, each focused on a particular function in the system:

- Administration: All administrative information can be found under this section.
- Shipping/Receiving: This dialog concerns itself with all shipping and receiving matters at the store. It includes, warehouse, store and supplier deliveries as well as returns to these entities.
- Inventory Management focuses on all the elements that can affect inventory positions within the store (excluding point-of-sale).
- Lookups: Under this dialog the user can find detailed information regarding items, suppliers and containers.
- Reports: This function will call up the reporting tool associated with SIM allowing the user to print custom and base reports.

Most of the inventory features discussed in this section are applicable to both Oracle Retail Mobile Store Inventory Management (the handheld devices) and the Oracle Retail Store Inventory product (the PC).

The handheld device enables the user to register items for the different inventory transactions with more accuracy by scanning barcodes and validating transactions against centrally deployed reference data.

Inventory Adjustments Functional Overview

To assist in maintaining perpetual inventory, SIM provides the ability to create inventory adjustments for all items within a store. SIM conveys changes to the merchandising system. Inventory adjustment functionality within the SIM system can be accomplished on a PC-based deployment, on a wireless handheld device, or on a combination of the two deployment methods.

Inventory adjustment processing within SIM includes the following features:

- Reason codes, which correspond to dispositions, can be assigned to inventory adjustments, thus moving stock to various inventory buckets. This code not only is used for reporting purposes, but also indicates to the system whether the amount is to be incremented or decremented and in which direction. Two examples follow:
  - Example 1:

    A reason code of Removed for repair would indicate to the system that the inventory for the selected item is to be decremented from the available stock on hand (SOH) bucket and incremented in the unavailable SOH bucket.
  - Example 2:

    A reason code of Return from repair instructs the system to move the selected inventory by decrementing the unavailable SOH bucket and incrementing the available SOH bucket.
- Stock on Hand and Unavailable inventory are not only used to indicate to a store user what is available to sell, but it also ensures proper replenishment ordering for all the stores through the central replenishment system.
- Manual or automatic adjustments can be made to the SOH inventory level for an item.
- Automatic inventory adjustments have hard coded reason codes that cannot be changed. These need to be set up in an identical manner in RMS. An example of these hard coded values is reason code 76-79 which is used to reverse stock count inventory adjustments that are caused by late sales.
- Moving stock to an unavailable bucket creates a pending inventory adjustment record. Later, these pending adjustments can be transformed into inventory adjustments out of inventory or back into available inventory.
- Receiving damaged inventory will automatically add quantity to the unavailable pending inventory adjustment record.
- The customer order (inventory reservation) requests will either increase or decrease the unavailable quantity by adjusting the customer reserve quantity bucket.
- System used inventory adjustment codes can be displayed to or hidden from the user.
- The system notifies the merchandise system of all inventory adjustments.
- Serialization support:
  - Users are allowed to move serialized inventory into unavailable inventory positions (pending inventory adjustment record), or out of unavailable.
  - When using auto generated UINs, it is possible to add new serialized inventory into the store when selecting a disposition that would normally increase SOH.
  - Moving serialized inventory out of inventory is permitted.

A Summary of Reason Codes and Dispositions

The following table (shown for example purposes) provides a list of SIM's reason codes that are preloaded, their descriptions and the dispositions that are linked to the reason code.

For example, code 83 refers to a theft, which indicates that the stock is moved from available in the store to out (the stock is gone from the store).

Note: Some reason codes are flagged as system-required. Do not remove these reason codes. If they are removed, SIM will not function properly.

Table 4–1 Preloaded Reason Codes

Internal Code	External Code	Reason Code Description	Disposition
1	1	Shrinkage	Available -> Out
2	81	Damage - Out	Available -> Out
3	82	Damage - Hold	Available -> Unavailable
4	83	Theft	Available -> Out
5	84	Store Use	Available -> Out
6	85	Repair - Out	Available -> Unavailable
7	3	Repair - In	Unavailable -> Available
8	86	Charity	Available -> Out
9	87	Stock In	Out -> Available
10	88	Stock Out	Available -> Out
11	89	Dispose from on Hold	Unavailable -> Out
12	90	Dispose from SOH	Available -> Out
13	91	Stock - Hold	Available -> Unavailable
14	92	Admin	Available -> Out
15	93	Store Customer Return	Out -> Available
16	94	Product Transformation – In	Out -> Available
17	98	Product Transformation – Out	Available -> Out
18	95	Consignment	Available -> Out
19	96	Ready to Sell	Unavailable -> Available
20	97	Returns	Unavailable -> Available
24	77	Unit Late Sales Decrease SOH	Available -> Out
25	79	Unit and Amount Late Sales Decrease SOH	Available -> Out
26	78	Unit and Amount Late Sales Increase SOH	Out -> Available
27	76	Unit Late Sales Increase SOH	Out -> Available
28	97	Return to Unavailable	Available -> Unavailable

Table 4–1 (Cont.) Preloaded Reason Codes

Internal Code	External Code	Reason Code Description	Disposition
29	180	Customer Order Reservations - In	Available To Sell -> Customer Order Reserve
30	181	Customer Order Reservations - Out	Customer Order Reserve -> Available To Sell
31	75	Stock Count UIN Unavailable to Missing	Unavailable -> Available

Updating Reason Codes

SIM provides a UI which allows the user to add new reason codes to synchronize with those setup in RMS. It is also possible to hide them from users, and indicate which are system controlled. A brief description indicates the disposition of the item for easier setup.

Inventory Adjustment Reason Maintenance UI will allow for Inventory Adjustment Reasons to be maintained in a UI versus directly through the database:

- This screen lists external code, description, disposition, use in UI and system required indicator of all the available reason codes in SIM.
- All the system required reason codes would have its system required indicator checked. Any reason that is marked as system required can not have its reason code, disposition and system required indicator changed through this UI. Only description and the use UI indicator are allowed to be modified.
- The user will not be allowed to delete the inventory adjustment reason codes that are marked as system required.

The following Reason codes are marked as System Required on install:

Table 4–2 System Required Reason Codes

External Code	Description	Disposition	Functional Area
1	Shrinkage	Stock On Hand	Wastage Batch
75	Stock Count Unavailable to Missing	Stock On Hand & Unavailable	UIN not counted during stock count
76	Unit Late Sales Increase SOH	Stock On Hand	Late Sales processing for unit counts
77	Unit Late Sales Decrease SOH	Stock On Hand	Late Sales processing for unit counts
78	Unit and Amount Late Sales Increase SOH	Stock On Hand	Late Sales processing for unit and amount counts
79	Unit and Amount Late Sales Decrease SOH	Stock On Hand	Late Sales processing for unit and amount counts
82	Damaged Hold	Unavailable	Transfers, Direct Delivery, Warehouse Delivery
87	Stock In	Stock On Hand	All stock counts
88	Stock Out	Stock On Hand	All stock counts
96	Ready to Sell	Unavailable	Transfers, Direct Delivery, Warehouse Delivery

Table 4–2 (Cont.) System Required Reason Codes

External Code	Description	Disposition	Functional Area
97	Returns	Unavailable	Returns (add unavailable sourced item/quantity to return)
145	Due to return to Vendor	Unavailable	Returns (remove unavailable sourced item/quantity from return)
180	Customer Order Reservation In	Customer Order Reserve	Customer Order Web Service
181	Customer Order Reservation Out	Customer Order Reserve	Customer Order Web Service
Pending-Reason Code	Unavailable	Unavailable	Used for pending Inventory Adjustment records

- Only records that have a checked value for the use in UI will be allowed for use in the inventory adjustment screen.
- The user will be allowed to add additional Inventory Adjustment Reason Codes, both system-required and non-system-required, through this UI.
- The change made through this UI is not integrated with RMS. Retailers have to make sure that the changes done in this screen are in sync with RMS.

Note: SIM and RMS must have the same inventory adjustment reason codes to work properly, with the exception of the Pending Reason Code, which is used for internal purposes only.

Figure 4–1 Business Process Flow – Inventory Adjustments PC

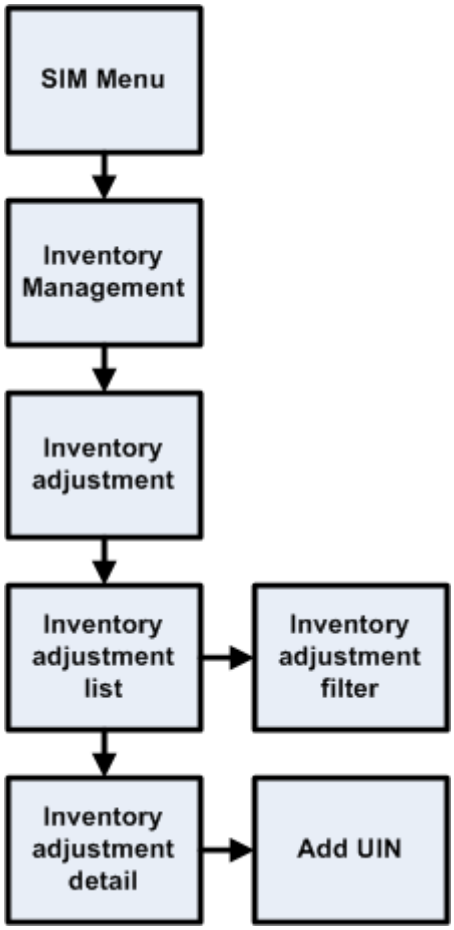
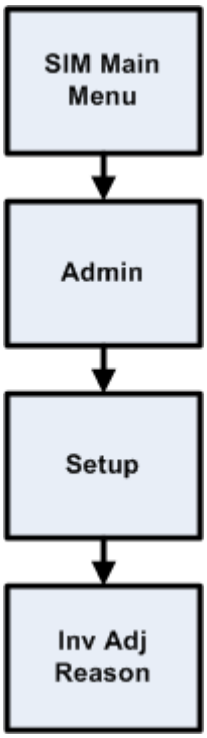


Figure 4–2 Business Process Flow - Inventory Adjustment Reason Maintenance PC



Wastage Functional Overview

Wastage is the process through which inventory is lost over time (bananas turning black, for example).

In order to maintain more accurate inventory values, SIM uses two methods to control wastage:

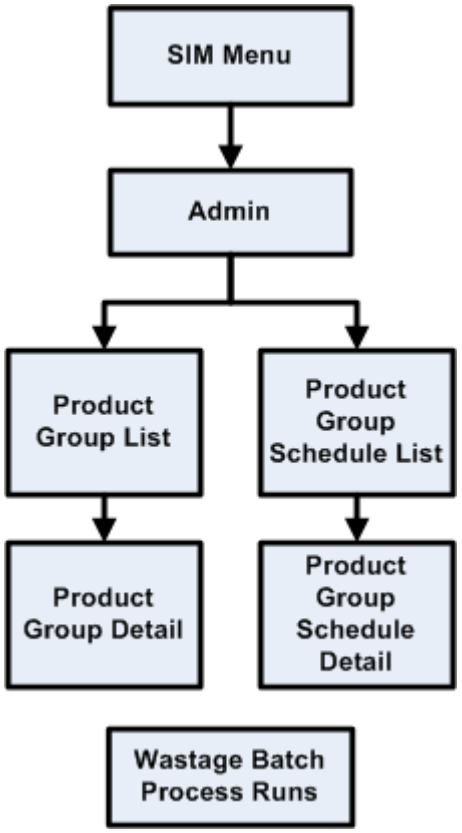
- The first method provides users in stores the ability to create wastage product groups. Variance percentage or standard UOM amounts can be set up on the wastage product group. Individual items and item hierarchies can be associated into a product group.

  A user can schedule the date when a wastage product group batch process is run, and inventory adjustments are automatically made based upon the variances setup on the product group. Inventory adjustments are sent over the Oracle Retail Integration Bus (RIB) to the merchandising system.

- The second method is controlled through the sales process. The external audit system provides a percentage or quantity in its sales upload file that indicates by how much the inventory needs to be reduced in addition to the sold quantity.

Each of these methods has a specific use or need. The first method is usually used when an item's size is reduced because of not being sold over time. For example, meat will become lighter as fluids evaporate. Other items, such as cheese or ham, will only be reduced when the outside layers are cut off to sell the item.

Figure 4–3 Business Process Flow (non-sale bases)



Store Orders Functional Overview

Store orders are used to create, change and approve orders to a supplier or transfers requests to a warehouse. When there is a shortage of items, or demand for particular items increases, store users need to have the ability to create store orders. The user selects either a warehouse or a supplier and adds the items and quantities. The store orders use Oracle Retail Service Layer (RSL) to action the order in RMS.

Store ordering functionality is very similar to item request functionality. Unlike item request functionality, which is only valid for items that are on the store order replenishment process, store order functionality is valid for all items, cannot be scheduled and is only available on the PC.

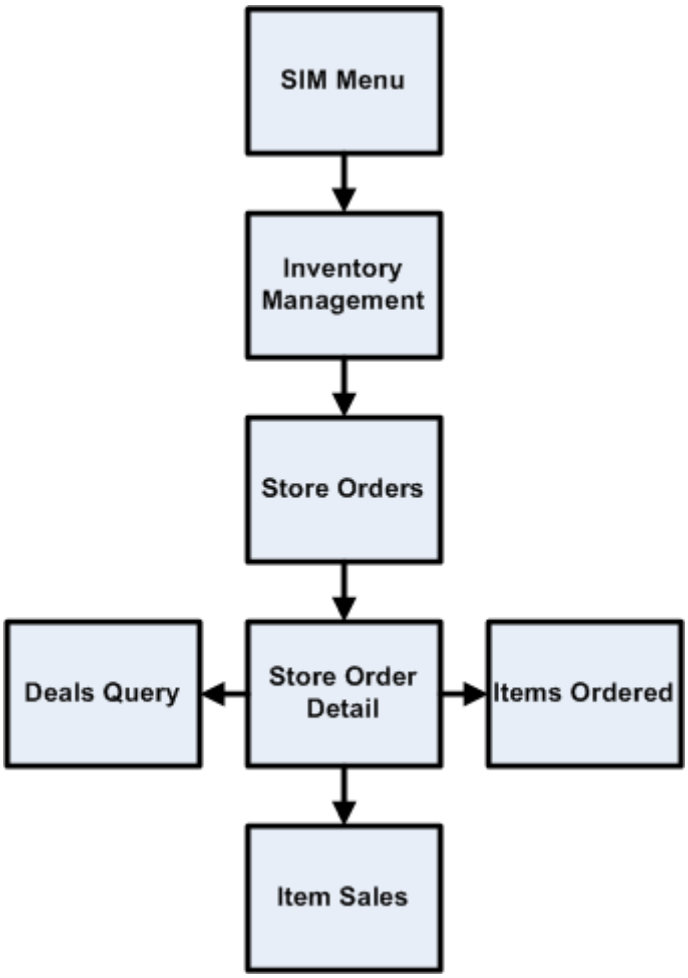
Store Orders also create the purchase order or the warehouse transfer immediately, while Item Requests depend on replenishment attributes and the nightly batch run from RMS.

Store order processing within SIM includes the following features:

- Create orders for the supplier or the warehouse.
- Save the creation of the order without approving it.
- Amend items and orders in SIM that were created either manually or through replenishment in RMS.
- Delete pending orders.
- Approve store orders.
- Query off-invoice deals when editing an existing Store Order to a Supplier.

- Query item's sales and store orders when editing an existing Store Order.

Figure 4–4 Store Orders Business Process Flow – PC



Item Requests

The item request functionality gives the user the ability to request inventory for individual items using the replenishment and sourcing parameters of the merchandising system (RMS) from within the SIM application directly. This functionality empowers the store by giving the store user the ability to manage stock shortages and increased demand using the SIM application.

This functionality differs from that of store orders. In terms of item requests, SIM sends a request to the merchandising system that is generally processed using the merchandising system replenishment and sourcing parameters. Store orders functionality, on the other hand, allows the user direct access to the merchandising system (RMS) and does not enforce the replenishment or sourcing parameters of the merchandising system.

A SIM user is able to use item request functionality to request items regardless of the replenishment type normally used by the merchandising system to replenish the item.

All items are sourced from either a warehouse or through a supplier depending on the sourcing parameters for the item specified in the merchandising system. Items specified as using Store Order replenishment (or items that are not set up for auto-replenishment at all) are sourced through the creation of one-off purchase orders or warehouse transfer requests only after the store has requested inventory using the Item Request functionality.

Any quantities requested for items that have a replenishment type other than Store Order are added above and beyond the quantity that is normally sourced through the merchandising system on the item's next replenishment review date. However, if the requested delivery date falls prior to such an item's next replenishment review date, the request is sourced through the creation of a one-off purchase order or warehouse delivery request instead. All inventory requested is sourced to the store at the earliest possible date given the replenishment review date, the supplier or warehouse lead time, and any other factors that may influence the time it takes a delivery to reach the store.

For store order replenishment items, the user can specify a time slot during the day by which the ordered items are to be delivered to the store. This is useful for ordering breakfast items early morning (doughnuts), lunch items (sandwiches), and hot meals for the evening. Item requests can be created with timeslot deliveries for as early as today's date. Security exists on the delivery timeslot fields, at both the store level and the user level. The user cannot delete line items or the master item request if the user does not have data permissions for a selected timeslot.

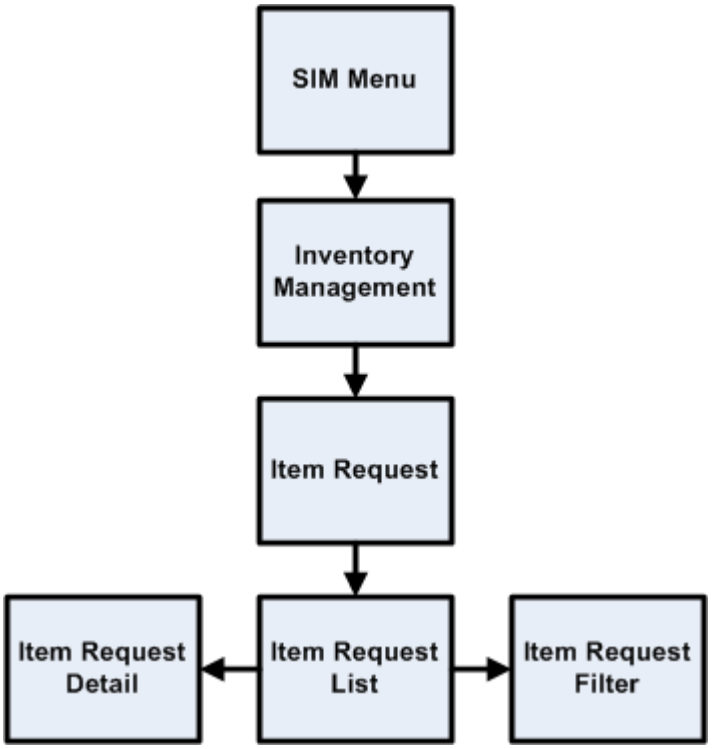
In addition to being able to manually create Item Requests, the SIM user is able to schedule Item Requests for review through front-end product group screens on a cyclical basis. This functionality facilitates the request of items that are specified as using Store Order replenishment by allowing the user to add individual items, as well as entire sections of the merchandise hierarchy, to an Item Request Product Group. When the Item Request Product Group is scheduled for review, SIM automatically generates a blank Item Request and adds all items within the specified merchandise hierarchies that have a **Store Order** replenishment type to the Item Request, along with any individual items specified as part of the Item Request Product Group. The user can then enter the actual quantities of the items necessary, and submit the request. Note that the user also has the ability to add items that do not have a **Store Order** replenishment type to an Item Request Product Group, but only on an individual item-by-item basis.

The following list summarizes the Item Request functionality that is available in SIM. Because of this functionality, the SIM user has the ability:

- To create an item request product group and schedule it for review.
- To manually create an unscheduled item request.
- To search for and view an item request whether created manually by a user or automatically by the product group scheduler.
- To edit a pending item request.
- To delete a pending item request.
- To request a pending item request.
- To save changes to a Pending Item Request without requesting it.
- To print an Item Request Report.

The SIM database contains a view called the Item_Request_Report_V that contains all of the data for this report.

Figure 4–5 Item Requests Business Process Flow – PC



Price Changes Functional Overview

The retailer uses price changes to change the price of a particular item at a location. Price changes are performed only on the PC.

In the merchandising system (such as RMS), users create the initial retail prices for items that will flow into SIM. After the initial prices have been set, ensuing control of prices is handled through a price management system. The price management system uses price zone structures or different levels to ensure consistent pricing within an area. Regardless of the level at which the initial prices are set, all prices in RMS and SIM are held at the item/location level.

Once users are managing prices in a price management system, they can use a flexible structure to control the retail prices through permanent, promotion or clearance changes. This functionality allows the user to use different sets of locations to control the retail prices of items without being locked into a zone structure. As a result of this flexibility, all prices are held at the item/location level, while they can be managed at higher levels.

In RMS, an indicator at the item/location level determines whether SIM users can request changes to an item's retail price at a specific location. This indicator is editable and controls behavior going forward but not in the past.

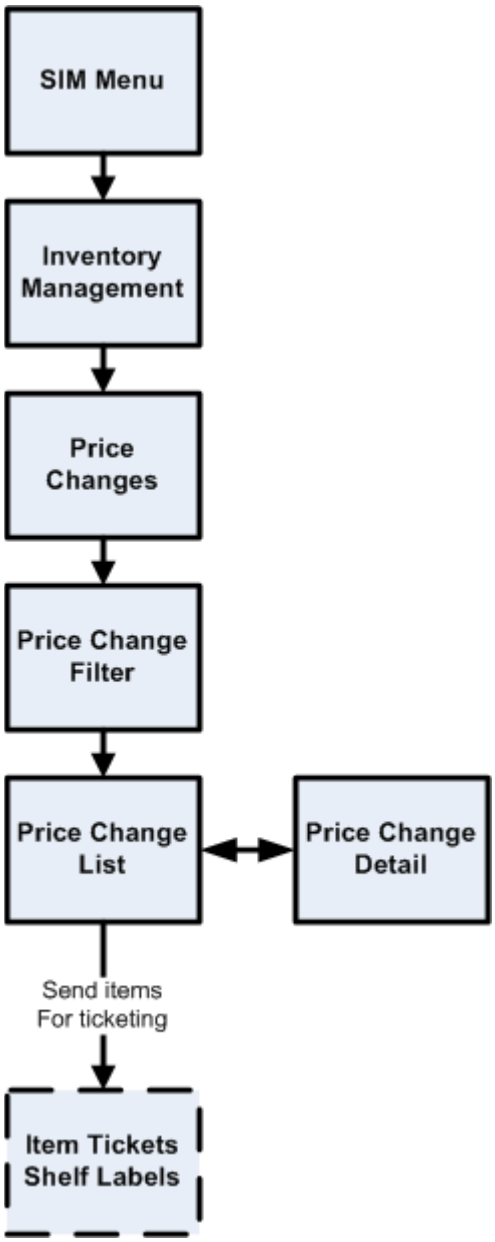
If SIM users have control of the retail price for an item at a location, they are able to send price change requests for permanent price changes, clearances or simple promotions to a price management system in real time. The price management system checks for any conflicts and provides a response to SIM regarding the status of the request. If the request is accepted, the price management system also sends a price change event back to SIM.

SIM users are able to edit and create price events given the following assumptions and restrictions:

- The item/location store control-pricing indicator must be set to **Y** for an item.
- The price event must not be a complex promotion (such as multiple promotions or regular price changes on the same day, buy/get, threshold, min/max).
- Any change requests, if approved, update the existing price event in the price management system.
- SIM can modify the retail price and effective dates for a price event. If a user requests a new price change for the same item/location/date instead of changing/correcting the existing price event, the request is sent to a price management system as a new event request. It undergoes conflict checking in the price management system. If any conflicts are found with existing price events, a rejected response is communicated to SIM.
- If an item/location is not set up for store controlled pricing in RMS, SIM users view all price events that are sent from a price management system, but they have no control over them. SIM is unable to create new price change requests to be sent to a price management system.
- Communication between SIM and a price management system is handled by RSL (providing a near real-time connection between the applications). Normal operation of pricing assumes that RSL and a price management application are both available. No manual override is provided within SIM.
- Prices are interfaced to SIM through the RIB or the bulk price batch.

It is recommended to only run one of the two interfaces at the time.

Figure 4–6 Price Changes Business Process Flow – PC



Ticketing Functional Overview

Tickets and labels can be generated from price changes, item description changes and from purchase orders (PO) that have been received.

SIM allows stores to print shelf edge labels and item tickets for stock.

Item tickets and shelf labels can be created and printed for individual items in the Item Tickets dialogs that exist on both the PC and the wireless device. Items in the ticket-printing list can be filtered by:

- Hierarchy
- PO
- Ticket type

- Label type
- Promotion ID
- From and to effective dates

Multiple items can be selected to be printed at once.

Tickets can be created on the PC in the following ways:

1. Manual:
    a. Individual ticket—When creating an item ticket, the user provides the quantity of the item to print and an override price (if necessary). The override price in ticketing is used to indicate the old price or a special promotion allowing the user to show the mark down.

    Note: This override price does not generate a price change.

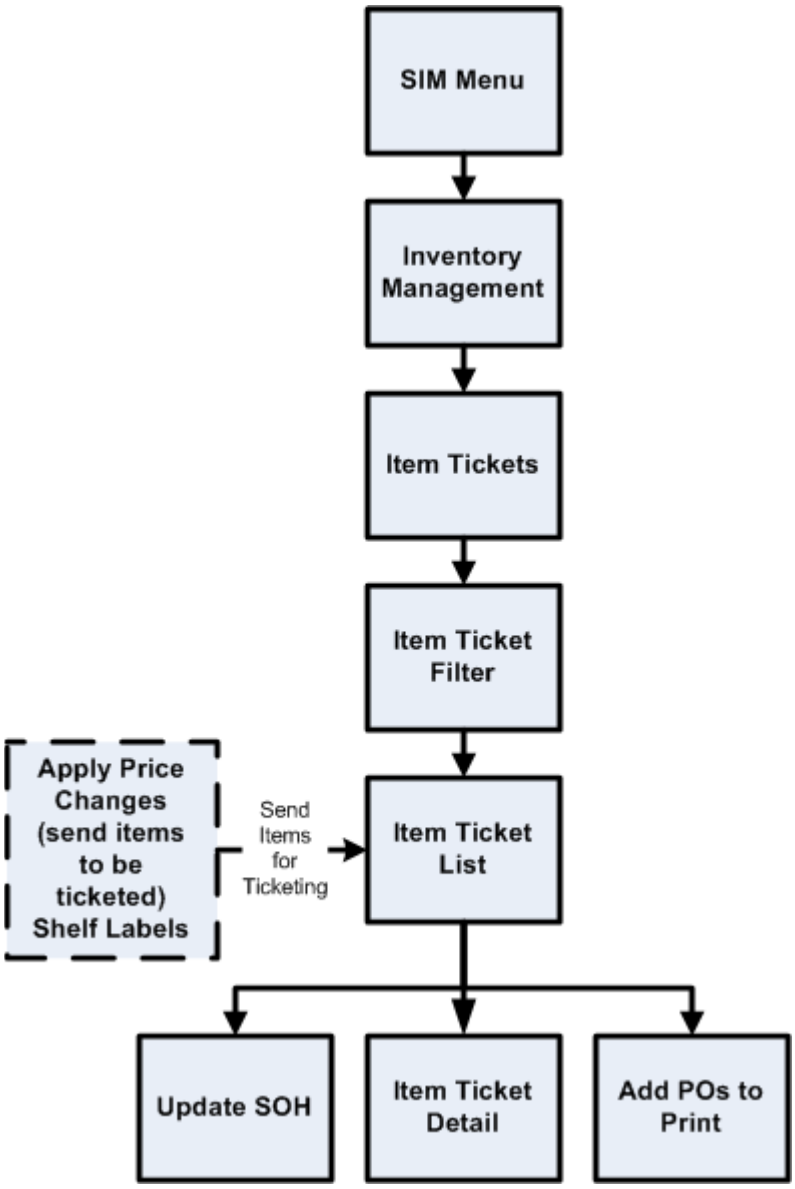
    b. Pricing dialog —Users can also send tickets to the Item tickets dialog to print at a later time by selecting price changes from the price change list screen and then and having it added to the Item Tickets dialog.

    c. PO receipts —Users can also create item tickets for purchase orders that have been received: the purchase order is selected and the corresponding received shipment on the Add PO screen is accessed from the Item Tickets. Item Tickets would then be generated for the items on the purchase order for the received quantities.
2. Automatic:
    a. Item description changes
    b. Price changes received from an external system
    c. UDA changes for an item

The handheld can only print manual individual created tickets. It is possible to use belt printers as long as they have their own unique printer network ID.

Label formats and label quantities will be maintained at the micro sequence location level for items setup in sequencing. This allows for defaulting label types based on primary location.

Figure 4–7 Ticketing Business Process Flow – PC



Ticketing UIN Support

SIM automatically prints tickets when a serial number is auto-generated. A ticket for the AGSN can also be printed using the Item ticketing dialog. Serial numbers cannot be printed.

Sequencing Functional Overview

Sequencing functionality provides users the ability to know the relative location of an item in a store. Sequencing a store improves store processes and reduces the time that employees spend looking for items (during a stock count, for example). The retailer can sequence all items in the store and create unique locations to hold the items. The system can prompt users to a specific location to look for a specific item. Sequencing functionality within the SIM system can be accomplished on a PC based deployment, on a wireless handheld device, or on a combination of the two deployment methods.

Sequencing functionality includes two means by which items can be assigned to places within a store: Macro sequencing and micro sequencing. When ordering, the system follows this pattern.

Macro sequences represent the highest level of locations that are set up in the store. The user can create macro locations, assign items to macro locations, remove items from macro locations, move items within macro locations and re-sequence an entire macro location (wireless only).

A micro sequence is the lowest (most granular) item location level.

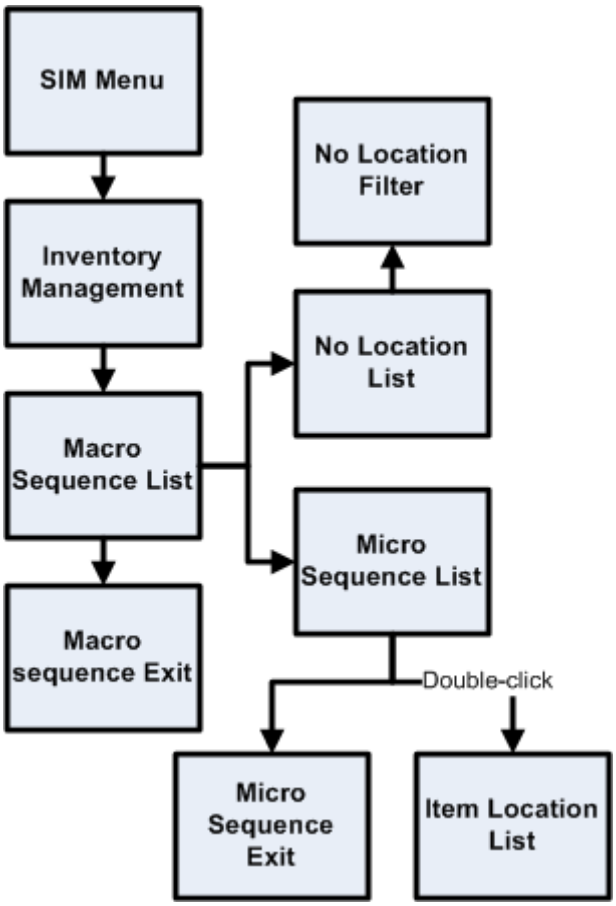
The following table provides an example of sequencing:

Table 4–3 Micro Versus Macro Sequencing

Macro Sequence	Micro Sequence
Produce	Oranges
	Apples
	Bananas
	Oranges
Frozen foods	TV dinners
	Burritos
	Burritos
Cereal	Toasted oats

Sequencing is used within Stock Counts and Shelf Replenishment to aid the user in proceeding to the next item during a count.

Figure 4–8 Sequencing Business Process Flow – PC



Shelf Replenishment (Pick Lists) Functional Overview

The replenishment process attempts to ensure that the shop floor inventory is set at a level best suited for customers.

Shelf replenishment functionality within SIM is related to the movement of goods from the back room to the shop floor. For example, when a user sees that a certain soda quantity is low, he or she can instigate a replenishment process so that more of the soda is moved from the back room.

Shelf replenishment-related processing within SIM includes the following features:

- The system calculates what should be held on the shop floor to ensure that customers' expectations of availability are maintained. Store employees are driven to replenish the most urgently needed items first.
- The system offers a display of the location from which stock can be picked to ease the search for stock when the pick lists are being filled.
- The system allows picking requests to be generated on demand.
- The system leads the user around the back room in micro sequence order, so that items can be picked in the most efficient matter.

Replenishment requires that the available SOH be divided into three buckets: shop floor, back room, and delivery bay. Because of these buckets, shelf replenishment affects almost every area in the application. For example, inventory adjustments and transfers are affected because the system must take the inventory buckets into account when engaging in these areas of functional processing. The system's available inventory is the sum of the three buckets.

When merchandise becomes available (enters the store through a transfer, a DSD, and so on), the merchandise is always placed in the back room bucket.

The user can create a within-day or an end-of-day pick list. The two different types of pick lists have store level configurations for the fill level. Typically an end-of-day pick list would have a higher fill level than a within-day pick list, as there would be more time to stock the shelves.

When the user creates a pick list, the system runs a replenishment calculation that checks for those items that belong to pick list product groups. The system takes those items and compares their capacity to their shop floor SOH. The system then generates a pick list in order of the items that need replenishment the most and orders them in sequential order for the user. For within-day pick lists the system will stop when the amount to pick is equal to the amount suggested by the system. For end-of-day pick lists the system continues until all items that need picking are picked.

Pick lists can be created on the PC or the handheld and can be fulfilled on either device. If they are created on the PC, the user is able to action them on the handheld.

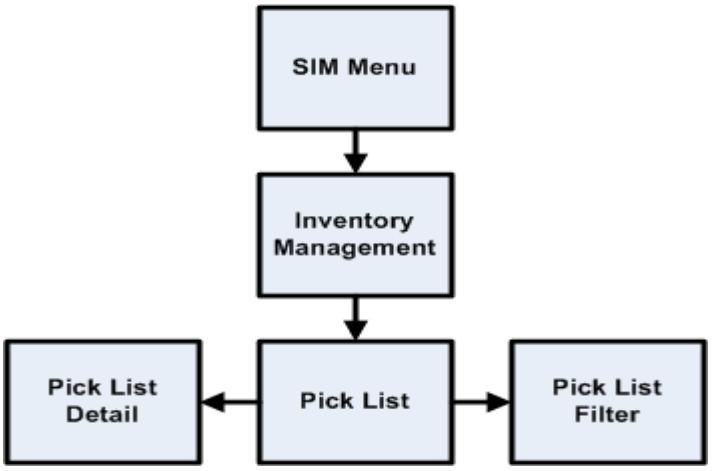
Replenishment Calculation Summary

Once the calculation is started, the system engages in the following processing:

- The system gets all the items that are sequenced on the shop floor with capacity in the product group.
- If a previous pick list exists for the selected group and it is in progress, the system assumes that all of the items on the pick list will be completed. If a previous pick list exists and is not started, the system deletes the old pick list and creates a new one.
- The system checks and gets the configuration parameters established through the GUI (group unit of measure, fill percentage, and so on).
- The system gets the shelf quantities and available SOH for the items found.
- The system converts the quantities to the correct group default unit of measure.
- The system compares the shelf quantity to the summed shop floor capacity for every item to determine the percentage the item is out-of-stock.
- Once the out-of-stock percentage is calculated for every item, the system orders the items from the highest out-of-stock percentage to the lowest. If any of the items have the same out-of-stock percentage, the system uses the item that has the least amount on the shelf. For example, if item A has 10 out of 100 on the shelf, and item B has 1 out of 10 on the shelf, they both have the same out-of-stock percentage. However, the system considers item B a higher priority because there is less of it on the shelf.
- For each item, the system calculates the pick amount that should be brought from the back room/delivery bay to the shop floor. Keeping the items in priority order, the system looks at the available SOH, the items in the back room/delivery bay, and to what percentage the shop floor needs to be filled. The system takes the inventory from the back room first and then takes the inventory from the delivery bay. The shop floor quantity can only be equal or less than the capacity.

- If the pick list type is within day, the system stops when the amount to pick is equal to the summed pick amount calculated by the system.
- If the pick list type is end of day, the system continues until all of the items are completed.
- If the system generated pick amount is a decimal, the system rounds down to the nearest whole number.
- The system generates and displays the list in sequence order to the user. If the items are not sequenced in the back room, the system displays the items in item ID order.

Figure 4–9 Shelf Replenishment Business Process Flow – PC



Stock Counts Functional Overview

SIM provides the ability to schedule, perform, and authorize stock counts. SIM includes the following types of stock counts, each of which is described in this section:

- Ad hoc
- Unit
- Unit and Amount
- Problem line

> **Note:** The counting processing is identical among the unit only, unit and amount, and problem line stock count types. What differs among these three stock count types is either the setup or the items being counted.

SIM includes the following types of counting methods for stock counts, each of which is described in this section:

- Un-guided
- Guided
- Third Party

Portions of the stock count functionality, such as the setup of product groups, schedules, and authorizations, are performed on the PC only. The actual counting of inventory can be performed on both the PC and the wireless device. A master count is created for a single product group and is then broken down into one or more child counts based on the counting method or hierarchy breakdown chosen during the product group setup. The user is able to save the child stock count on the PC or handheld and resume at a later time.

Future Stock Counts

Future stock counts are stock counts for which the scheduled date has not yet arrived. The user can view a list of future stock counts, and the user also has the option to generate a future count to view the count details. The user cannot take any action on a future stock count, the purpose is to allow the store to view future workload so that the store can plan ahead for staffing.

Unscheduled Counts – Ad Hoc Stock Counts

An ad hoc stock count is performed by a user walking through the store scanning any items that need to be counted. They have no group or schedule functionality associated with them.

Ad hoc stock count processing within SIM includes the following features:

- The system creates an inventory snapshot as each item is identified and added to the stock count.
- Ad hoc counts can only be initiated on the handheld. Once the count has been saved on the handheld, it can be completed later on the handheld or PC; any existing ad hoc stock count can be retrieved and resumed on the handheld and PC.
- The system utilizes discrepancy thresholds (established in administration setup by the retailer) based on a percentage or standard unit of measure by the item's class in the merchandise hierarchy.
- On the PC, the system allows for the authorization of items that are discrepant (based on the percentage or standard unit of measure thresholds).
- Where the authorized item quantities entered by the user differ from the SOH, the system creates inventory adjustments that are sent to the merchandising system. See "Inventory Adjustments Functional Overview" in this chapter.
- It is possible to have multiple users scan for the same ad hoc stock count.
- Since the user determines the order the items are scanned in, and no predefined list exists, sequencing has no impact to these counts.

Scheduled Stock Counts

Several different scheduled stock counts exist, each with their own specific differences. They do however have common setup pieces.

- Individual items or item hierarchies are associated into a single unit product group for the purpose of scheduling a stock count.
- Stock Count product groups can be scheduled for a stock count on a specified day or on scheduled intervals (for example, daily, weekly, monthly, or annually).
- One or more stores can be assigned to the scheduled stock count, and each store will complete their stock count individually.
- A group of valid stock count items is generated at the store level using batch processes.

- Users with proper security are prompted of any discrepancies outside of set tolerances, and the system can automatically force a recount if the discrepancies are too high. The system utilizes discrepancy thresholds (established in administration setup by the retailer) based on a percentage or standard unit of measure by the item's class in the merchandise hierarchy.
- An auto-authorization feature can be selected during the product group setup. If set up for auto authorization, SIM automatically authorizes the stock count after the count or re-count has been completed. In this case, no user intervention is needed to confirm the count. For Sarbanes-Oxley Act or auditing process requirements, this can be the optimal way to process a unit and amount count.
- Retailers can perform their Unit, Problem Line or Unit and Amount stock counts using different counting methods:

Third Party Stock Counts These stock counts are scheduled in SIM, but the actual counting process is performed using the third-party system. Once the physical stock counting process has been completed, the third-party system exports the results of the count to SIM.

SIM compares the count information with the stock-on-hand (SOH) value currently held in SIM. In SIM, users can view all items in the stock count that are discrepant and non-discrepant when compared to the SOH figure. Pre-defined variance limits (units, percent, and value difference) are used to determine which items fall outside the acceptable level and are therefore considered discrepant.

Any items SIM does not recognize can be added through the Rejected Items dialog. Items that require UINs can also be assigned UINs through the Rejected Items dialog.

The items associated to a product group will be associated to a master stock count when generated, which in turn can be broken down based on the hierarchy selected by the user during the product group setup (for example Department, Class, Sub-class).

It is possible to create Third Party Stock Counts for any merchandise hierarchy level supported in SIM.

For Third Party Stock Counts to work, the third party counter needs to receive an extract of the items, quantities and UINs to count from SIM .

This report can be printed to the screen and subsequently saved by a user. The report only needs to be accessible in the stock count dialogue.

The Third Party Stock Count Extraction Report allows a retailer to extract the snapshot value and in store UINs in an XML format that should be counted for a specific stock count. This report may need to be modified by individual retailers for different third parties when performing a stock count, and is only meant as a starting point.

If this report is generated before a snapshot is taken, the UIN and snapshot quantity field will be empty, so the assumption is that the extract will be created after the snapshot is taken.

Note: This will most often be used to print reports for third party stock counts, however it should work for all stock count types.

This XML report contains the following information:

- Header:
  - Stock Count ID
  - Store ID
- Detail
  - Item number (SIM SKU number)
  - Item description
  - Total snapshot quantity for the item
  - UINs for the item

Unguided Stock Counts When performing a scheduled stock count (unit or unit and amount), the user is able to scan items on the handheld without being prompted for which item to scan. This feature is controlled through the product group setup by choosing Unguided for the counting method. Multiple employees are able to scan items for the same stock count. This feature is controlled through a system option.

Unguided stock counts cut down the time it takes to completely scan a count and provides more flexibility. This process applies to both count and recount processes.

The items associated to a product group will be associated to a master stock count when generated, which in turn can be broken down based on the hierarchy selected by the user during the product group setup (for example Department, Class, Sub-class).

For unguided counts, the retailer can configure SIM to save the item count automatically when the user moves on to count a different item. For example, if the user scans item A three times and then scans item B, the count value for item A is saved when scanning item B. This automatic save reduces the need for store personnel to remember to save count values manually and it allows longer scan times without interrupting the scanning of items for the count. Users might forget to save, and if the system experiences a problem, the data recorded up to that point will be saved if SIM is configured to do so.

Guided Stock Counts Guided stock counts prompt the user for the next item in sequence. This feature is available for scheduled Unit, Problem Line and Unit and Amount counts and is controlled through the product group setup by choosing Guided count method.

SIM generates a master count that is broken down by location. A Child count is created for every macro location and the user is prompted to scan the next item based on its location within the store. If an item exists in multiple locations, the user is warned if the item has not been counted in all locations.

If sequencing is not set up for the items, the user is prompted in item order.

It is also possible to have these counts automatically processed, ensuring no store interference. This automatic third-party process does not require any counting or approval from store personnel.

Unit-Only Stock Counts

Unit-only stock counts are usually small stock counts setup on a recurring pattern every few weeks or monthly. Unit-only stock count processing within SIM includes the following specific features:

- Setup of the stock count can be done at the item, merchandise hierarchy level.
- On the PC, the system allows for the authorization of items that are discrepant (based on the percentage or standard unit of measure thresholds) or non-discrepant, or both.
- Where the authorized item quantities entered by the user differ from the SOH, the system creates inventory adjustments that are sent to the merchandising system. See "Inventory Adjustments Functional Overview", in this chapter.

Unit and Amount Stock Counts

Unit and amount stock counts are usually only done once or twice a year. They are often required by law to be performed once a year and are done for the entire store or specific merchandise hierarchies. They give the retailer the ability to consolidate the actual counted values for merchandise and the booking numbers at year end.

Unit and Amount stock count processing within SIM includes the following features:

- Setup of the stock count can only be done at merchandise hierarchy level.
- Unit and amount counts are scheduled and counted within SIM and then sent to RMS. RMS accepts the unit variances and the user inputs the variance amounts in Central Office.
- Unit and amount product groups are scheduled for a specified day.

The stock count schedule for unit and amount stock counts is sent to the merchandising system anytime it is created, updated or deleted.

- One or more stores can be assigned to the scheduled stock count.
- A stock count item list is generated at the store level using a batch process that runs daily.
- On the PC, the system requires the authorization of all items (based on the value, percentage, or standard unit of measure thresholds).
- Upon completion of authorization, a flat file is sent to the merchandising system with a header that contains the stock count ID, stock count date, and the store number that executed the count. The file contains details for each item and the quantity counted. This information is then staged in the merchandising system and can be used for reporting purposes.

Note: The **Export Results** button has been removed and results will automatically be exported to the merchandising system upon confirmation of the last child count. The user no longer needs to manually export the results to the merchandising system using the GUI.

Stock Counts UIN Tracking

For items that require UINs, the user must capture the UIN when performing a stock count on the PC or handheld. The count quantity will always equal the number of UINs captured for the item.

The count/re-count stages only allow the user to count UINs that already exist in the store. If a UIN does not exist in the store, the UIN can be added during the Authorize stage. When the count is confirmed, the UIN is created for the current store and the status moves to In Stock.

For UINs that have a status of In Stock after the initial count but move to another status before the count is authorized, SIM does not update the UIN to In Stock upon confirmation. In order to achieve this, a snapshot of the status is taken at the time the snapshot is taken. The snapshot will always be taken at the beginning of the child count, including Unit and Amount counts.

For UINs that exist for the item in the current store but are not counted on the stock count, the status gets updated to Missing upon confirmation of the count.

For AGSN items, SIM requires the user to scan the UINs to capture the quantity as it does for regular serial numbers. During the authorization process, the user is able to adjust the authorized quantity to account for items that are missing UIN labels. The user can auto generate UINs during the authorization process by clicking **Auto Generate** from the UIN popup.

An audit record is created when the stock count is authorized.

Problem Line Stock Counts

This functionality gives stores the ability to create automated stock counts according to predefined criteria (for example, the retailer could decide to count all of the items that have negative SOH values).

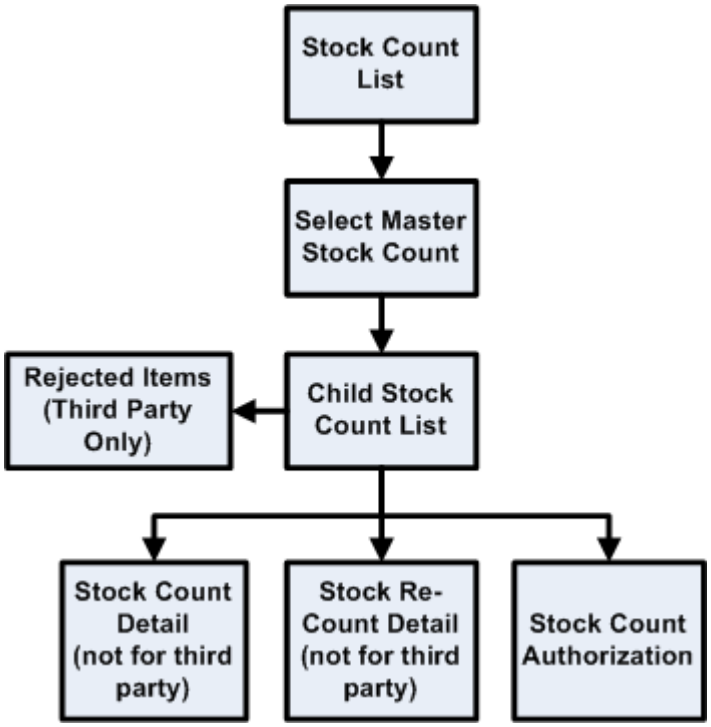
Once stores have established the criteria (based upon problematic areas), a batch process runs to find any items that meet the criteria. The found items are added to the scheduled stock count. They are counted in the same way as in a scheduled unit stock count. Note that problem line stock counts will be executed every day.

Problem line stock count processing within SIM includes the following features:

- Individual items and item hierarchies are associated into a single problem line product group for the purpose of scheduling a stock count.
- Problem Line product groups schedules will be defaulted to daily, every 1 day and cannot be changed
- On the PC, the system allows for the authorization of items that are discrepant (based on the percentage or standard unit of measure thresholds) and non-discrepant.
- Where the authorized item quantities entered by the user differ from the SOH, the system creates inventory adjustments that are sent to the merchandising system. See "Inventory Adjustments Functional Overview", in this chapter.

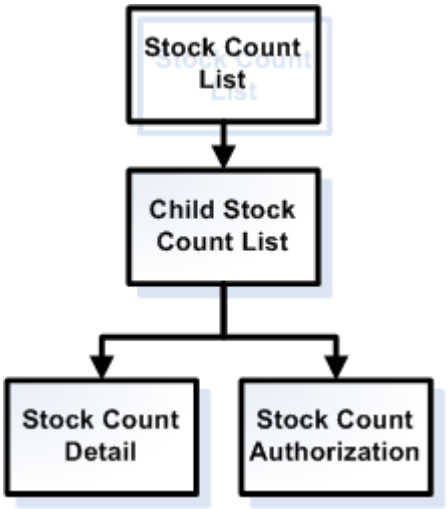
Note: With the exception of the extraction criteria, the execution (counting, snapshot taking, authorization) is the same as with a regular unit count.

Figure 4–10 Business Flow (Unit, Problem Line, Unit and Amount and Third Party)



Note: Third Party count/re-count process is not performed in SIM, it is performed through the third party system.

Figure 4–11 Business Flow – Ad Hoc (PC only)



Note: Ad hoc count process is initiated on the handheld by scanning an item. Once the count has been saved on the handheld, the user is allowed to complete the count from the handheld or PC. There is no re-count for Ad Hoc and an Ad Hoc will only have one child count. Authorization occurs on the PC only.

Figure 4–12 Business flow – Third Party

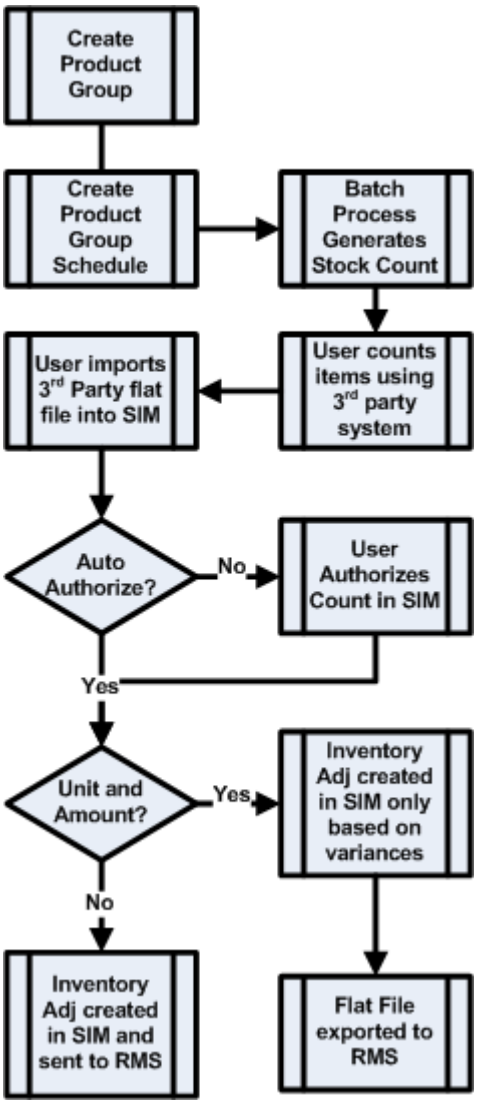
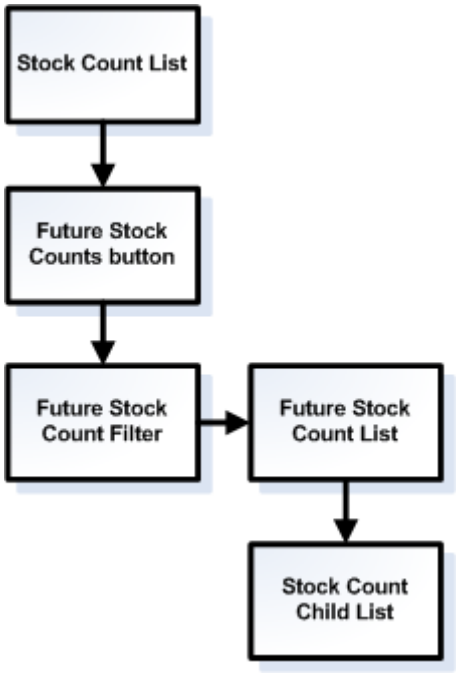


Figure 4–13 Business Flow - Future Stock Count



Item Basket

Item Basket functionality allows the user on the handheld to scan a list of items. This list of items can be interfaced to other applications through a web service to aid them in specific tasks. These tasks can range from line busting (Oracle Retail Point-of-Service), using it for wedding list generation or simply identifying trouble items.

Specific features include:

- Identifying a specific type of basket
- Scanning or entering quantity for the item
- Entering or auto generating a unique ID for calling it up
- Edit, delete and add functionality
- Print ticket functionality for register scanning

The following is an example of a possible business flow for line busting:

Items are scanned into the Store Inventory Management handheld and the basket is created in Store Inventory Management. Optionally, the customer can be presented with a printed ticket containing a barcode, which can then be presented at checkout.

At an Oracle Retail Point-of-Service terminal, the operator must enter the unique Item Basket ID or scan the barcode from the ticket printed by the Store Inventory Management handheld, and the basket details are retrieved from Store Inventory Management and displayed on Oracle Retail Point-of-Service for item tender.

If a UIN needs to be captured for an item, Oracle Retail Point-of-Service will have this responsibility.

Shipping and Receiving Functional Overview

SIM has four distinctive shipping and receiving dialogs:

- Transfers: Store-to-store transfer requests, dispatch, and receiving.
- Returns: Warehouse and supplier returns and return requests and dispatch.
- Direct store delivery (DSD): Supplier deliveries and Quick Order Creation
- Warehouse delivery

Store-to-Store Transfer Functional Overview

Within SIM, the following areas of functionality are related to transfers and are discussed in this section:

- The creation of store to store transfers
- The receipt of store-to-store transfers
- E-mail alerts

Store-to-Store Transfers

SIM allows for the lookup, creation, editing, and deletion of store-to-store transfers. A store-to-store transfer is the movement of stock from one store to another, within a given company.

This functionality can be performed on the PC deployment, on a handheld wireless device, or a combination of both. Users can create a transfer by selecting the receiving store and adding items by scanning and/or engaging in manual entry. The system verifies the receiving store is approved to receive the selected items and that the sending store has the available SOH inventory. A transfer can be immediately sent or saved to be dispatched at a later time. At the point the transfer is dispatched, SIM decrements the on hand inventory from the sending store and increments the in-transit inventory for the receiving store.

The following features of transfer-related functionality within SIM:

- Stock is differentiated by different buckets depending on stock status (for example, in-transit stock, reserved for transfer stock, and so on).
- The system automatically updates stock inventory on the basis of the status of the transfer.
- Buddy store functionality allows for the setup of a group of stores within a transfer zone in SIM to which the retailer often transfers items. This shortens the list of values that users select from when they create a transfer.

  Note: The retailer continues to have the option of creating a transfer to any store outside of the buddy store group, as long as it resides within the transfer zone.

- When saving a transfer before dispatch, SIM will communicate transfer positions to the external merchandise system and reserve internally the inventory. This allows for a more accurate replenishment.

An Overview of Stock Movement after a Successful Dispatch The stock moves from the transfer reserved and transfer expected buckets.

1. The transfer reserved quantity for the outbound location decreases as does the SOH for the outbound location.
2. The transfer expected for the receiving store results in a stock movement to the in transit bucket. The transfer quantity is removed from the in transit bucket to the SOH bucket when the receiving store receives the transfer.

Figure 4–14 Store to Store Transfers Business Process Flow – PC

Transfer Requests

Transfer requests provide stores the ability to request products from other stores or allow corporate users to move inventory across stores using the central merchandising system. Transfer Requests are accessed from the Transfer dialog. SIM allows for the lookup, creation, editing, and deletion of store-to-store transfer requests.

A store user creates a transfer request by first selecting the store to request the merchandise from and then adding items to the request. Once the request has been sent to that store, the user can either accept or reject the request. Once this is done, an e-mail is sent out to the requesting store to notify of the response. If the transfer request is rejected, inventory does not get updated. If the transfer request is accepted, the user is directed to the transfer create dialog where all of the items and quantities have been defaulted. From here on, the dialog will act as a regular transfer.

Retailers are only allowed to accept or reject a transfer request awaiting response on the PC. The actual transfer request can be created on either the PC or the wireless device.

Transfer Shipment

After a request has been approved, or during the creation a new transfer, the user is able to identify which units should be shipped. Through the context field, the user can indicate the purpose of the transfer.

A new transfer can be created on the PC or the handheld. SIM allows the creation, deletion and cancellation of a transfer.

If UINs need to be tracked for an item, the user must enter them instead of a quantity.

When saving a transfer, SIM reserves the inventory and communicates this information to RMS. This ensures inventory is not incorrectly appropriated for other means.

Transfers Receiving

The retailer is able to receive against transfers on both the handheld device and the PC.

On the PC, the store user can select the appropriate dispatched transfer coming into the users store to receive against.

On the handheld, the retailer can receive a store-to-store transfer by scanning an item on the transfer.

By scanning or manually entering, the user adds the items to be received at the transfer, item, or case level. The ability to receive unexpected items not originally on the delivery is configurable.

In the scenario where a transfer receipt being received is of substantial size and cannot be completed at one time, the user has the ability to save the delivery. This allows the user to save what has been received and return at a later time to continue receiving. Once the user has completed receipt of the entire transfer, the transfer would then be moved to a Received status. When the transfer is completed, SIM decrements the inventory from in-transit status and increments the on hand inventory appropriately. At this point, changes to the transfer receipt can no longer be made unless the system is configured to allow for receipt adjustments.

- During the receiving process, the store user has the opportunity to record any damaged or missing items on the transfer. An inventory adjustment record is written for damaged units (with a reason code of damaged-hold) to adjust the units from Available SOH to Unavailable SOH in the receiving store. This information is reported to the central merchandising system.

  Note: E-mails are also sent out to the sending store if a transfer received contained damaged items.

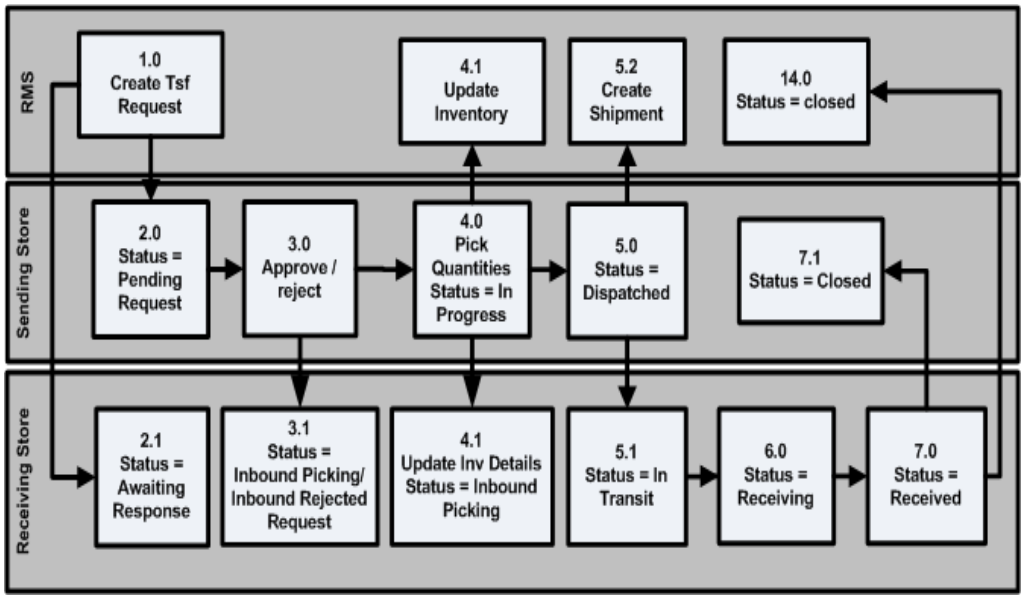
- When items are received for more than the dispatched quantity, the system adjusts the difference out of the sending store's SOH. No inventory adjustment record is sent to the merchandising system or displayed. An e-mail notification of the adjustment is sent to the sending store. The e-mail includes the transfer number, item numbers, and the quantities adjusted out.

- Depending on the settings, when items are received for less than the dispatched quantity, the system can adjust the difference in the sending store's SOH (no loss) or ignore the missing units (sending or receiving loss). No inventory adjustment record is sent to the merchandising system or displayed. An e-mail notification of the adjustment is sent to the sending store. The e-mail includes the transfer number, item numbers, and the quantities adjusted in.
- The inventory is not recognized in SIM until the transfer has been received.
- When receiving UINs, the user is restricted to the UINs that were shipped, or be able to add unexpected UINs depending on configuration.

  An auto receipt option exists for shipped transfers. This dialog can be found under the administration section, SIM Stores, Auto-Receive stores.
- In conjunction with the auto receipt store option, it is possible to auto-receive in full the moment an ASN transaction arrives or to auto-receive based on *x* number of days after the transfer has shipped. This is configurable through a system administration parameter.

> **Note:** E-mail Alerts: An e-mail alerts batch program will look at all of the dispatched transfers that have not yet been received within a configurable number of days and send out an alert to both the sending and receiving store. This batch is called TransfersOverdue.

Figure 4–15 Transfer Request to Transfer Receipt



The following table represents the different states a request or transfer can be in, based on the sending store or the receiving store.

For example, a transfer in dispatched status from the sending locations can be in **In Transit** or **Receiving** status in the receiving store.

Table 4–4 Different States of a Request or Transfer

Phase	Sending Store Status	Receiving Store Status
REQUEST	N/A	New Request
REQUEST	Pending Request	Awaiting Response
REQUEST	Outbound Rejected Requests	Inbound Rejected Requests
REQUEST	N/A	Cancelled Request
TRANSFER	In Progress	Inbound - Picking
TRANSFER	Dispatched	In Transit
TRANSFER	Dispatched	Receiving
TRANSFER	Closed	Received
TRANSFER	Cancelled Transfer	Inbound - Cancelled

Warehouse Delivery

Warehouse delivery functionality within SIM is utilized when goods are sent from a warehouse or external finisher to a receiving store. Warehouse delivery within the SIM system can be accomplished on a PC-based deployment, on a wireless handheld device, or on a combination of the two deployment methods.

SIM allows for receiving from any number of company-operated warehouses or external finishers. These entities must be approved shipping locations for the receiving store, and the items shipped must be approved for delivery to the receiving store.

When the transfer or allocation is created, SIM is able to display this information to the user with the estimated in store date.

The moment the warehouse or external finisher ships the transfer/allocation, SIM is notified over the RIB and moves inventory into an In-transit inventory bucket. When the user confirms the receipt of the ASN the inventory will be moved from In-transit to the Stock on hand (SOH). The merchandising system will be updated in near real-time with the received information.

Receiving can be done at the following levels:

- Advanced Shipping Notice – This receiving level assumes a retailer's distribution center or warehouse system is very accurate and the store accepts the entire ASN without checking the content.
- Container - Store users can scan the barcode on the container (pallet/distribution unit/carton) within a receipt to find the quantities contained within and receive all contents.
- Case/Item - This is the lowest level of the receiving process where each item is received individually. Additionally, an initial receipt can be done at a high level and saved to allow for detailed item level receiving at a later time. The ability to receive unexpected items not originally on the delivery is configurable.
- Warehouse Quick Receiving - Warehouse quick receiving allows the user to scan each container as it comes off the truck. The user can confirm and reconcile after all the containers have been scanned. This function is only available on the Hand Held.

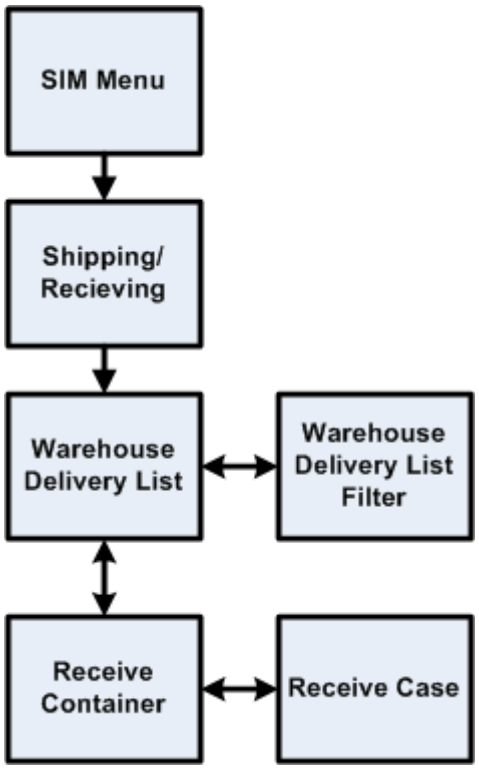
During the receiving process, the store user has the opportunity to record any damaged or missing items on the receipt. An inventory adjustment record is written for damaged units (with a reason code of damaged-hold) to adjust the units out of Available SOH and into the Unavailable SOH in the receiving store. This information is reported to the central merchandising system.

In the scenario where a delivery being received is of substantial size and cannot be completed at one time, the user has the ability to save the delivery as In Progress status. This allows the user to save what has been received and return at a later time to continue receiving. Once the user has completed receipt of the entire warehouse delivery it would then be moved to a Received status. At this point, changes to the delivery can no longer be made unless it is configured for Unit Receiver Adjustments.

When scanning a container that is listed as missing from a confirmed Advance Shipping Notice (ASN) in Quick Warehouse receiving on the handheld device, SIM receives the container instead of prompting the user with an error message. This receiving process follows the same logic as regular container receiving. The result of this operation is that the receipt is amended with the missing container now received. A message is sent to RMS to bring both systems in sync.

It is possible to auto-receive a warehouse delivery. External finishers and warehouse deliveries can be configured through a system administration parameter. The store user has the option to auto-receive in full the moment an ASN transaction arrives or auto-receive based on *x* number of days after the ETA date.

Figure 4–16 Warehouse Receiving Business Process Flow – PC



Warehouse Quick Receiving

Warehouse quick receiving allows the user to scan each container as it comes off the truck. The user can confirm and reconcile after all the containers have been scanned. This acts as a follow up audit after the truck has been unloaded and has left.

Quick Receiving with Missing Containers

When scanning a container that is listed as missing from a confirmed Advance Shipping Notice (ASN) in Quick Warehouse receiving on the handheld device, SIM receives the container instead of prompting the user with an error message. This receiving process follows the same logic as on the PC. The result of this operation is that the receipt is amended with the missing container now received. A message is sent to RMS to bring both systems in sync.

Warehouse Delivery UIN Tracking

When receiving containers using either the handheld or the PC, SIM validates the container to see if it contains any items that have a Capture Time of Store Receiving. If any items do, SIM requires the user to scan the individual UIN numbers within the Container.

If a serial number is not scanned, the quantity on the delivery will not increase. At this time, the ASN message from the warehouse does not include any UIN information.

For AGSN receiving, the user enters the received quantity and damaged quantity. UINs are not scanned during the receiving process for items that require AGSNs. SIM generates UINs and prints a ticket for each damaged and received item upon receipt confirmation. A business process should be put in place to put the AGSN tickets on the correct items.

Warehouse Quick Receiving allows for a user to quickly receive at the container level, therefore, if the container has items that require UINs, the user will be given the option to receive the UINs now or set the container aside to be received later.

When the delivery is accessed after it has been received, the user can view the UINs that were created.

External Finisher-Specific Logic

Through the returns dialogue, items can be shipped to an external finisher for repair. After the finisher has completed the repair work, a transaction is generated for SIM to receive against. The receipt is done in the warehouse delivery dialogue where the user can receive against the transaction, receive a single container or receive the individual item.

If the items were UIN items, the warehouse delivery dialogue will check to see if the UINs shipped to the finisher are being returned. If they are UINs that are new, the user can still receive them, but will be asked to confirm the delivery.

Direct Store Delivery (DSD)

DSD occurs when the supplier drops off merchandise directly in the retailer's store. This process is common in convenience and grocery stores, where suppliers routinely come to restock merchandise. In these cases, the invoice may or may not be given to the store (as opposed to being sent to corporate), and the invoice may or may not be paid for out of the register. The SIM system allows for the retailer to create new delivery records. This process allows the retailer to enter/scan a product bar code from any of the items in the delivery. Once the system verifies the bar code, the retailer can choose the supplier for the delivery. The system allows the retailer to either select an existing purchase order associated with that supplier to receive against, or to create a new purchase order. DSD delivery functionality within the SIM system can be accomplished on a PC based deployment, on a wireless handheld device, or on a combination of the two deployment methods.

The retailer can enter invoice information and receive items by the case/pack or by the item. The retailer can also print a delivery receipt once all items have been received, and the delivery is finalized. Note that the system is also able to handle deliveries partially received, allowing for multiple receipts against a single PO.

Through security permissions, the retailer can prevent users from over-receiving on the delivery. Once the expected quantity is reached, the user is prevented from receiving any more units. Receiving for damages can also be restricted through security. A system parameter allows the user to receive greater than the expected quantity but the stock on hand does not get updated for any units that exceed the expected quantity. A second system parameter allows the user to receive damaged units but, as with over-receiving, the stock on hand will not be adjusted for these units. These over-received and damaged units are moved to a separate table upon confirmation of the delivery and can be published to an external system. In case both parameters are setup, the damaged units will be removed before the undamaged units.

Upon completing the order, the system validates whether the supplier allows for:

- any discrepancies
- only overages, not short receipts
- no discrepancies based upon a supplier attribute

A security setting dictating whether supplier discrepancies can be overridden is checked against the receipt. If the discrepancies cannot be overridden, the receipt is not completed. The user has an option to reject the entire delivery. If overridden, the receipt process continues on.

Note: In a standalone environment, the supplier level indicator can be manually set up on the supplier in the SIM DB.

Upon completing the delivery, the SOH for the store is updated with the received quantities. An inventory adjustment record is written for damaged units (with a reason code of **damaged**) to adjust the units out of SOH in the receiving store.

The receipt and purchase order information is published to the RIB for the purposes of the merchandise system.

Depending upon system configurations, users can re-open direct deliveries and adjust received quantities (within an established number of days). Corrected data is then processed and resent to the merchandising system. This unit receiving adjustment functionality is only available on the PC.

If the unit cost configuration is turned on, in certain conditions, the user will be able to enter a unit cost for DSDs created in SIM. These costs will be used by the merchandise system to generate a Purchase Order(PO). When receiving against existing Purchase Orders, SIM cannot update the cost since it is controlled by the merchandise system. If it is not filled in, then the merchandise system will default the cost.

Receiving Against Advanced Shipment Notices (ASN)

Because of receiving-related processing within SIM, the retailer is able to receive against advanced shipment notices (ASN) on both the handheld device and the PC. ASNs that originate at the vendor are published to the RIB, and SIM subscribes to the data.

When a direct delivery is received, SIM checks for a corresponding open ASN against the PO.

Retailers are prompted as to whether they would like to apply the ASN to the delivery. If the ASN is applied, the shipped quantities from the ASN are applied to the quantity received for the direct delivery. Depending on configuration, if new items are included in the ASN but do not reside on the original PO, the items are added to the PO. Once the ASN is applied, the retailer can modify any of the received quantities.

A system option controls whether or not expected quantities are defaulted in for direct store deliveries with ASNs. This allows more detailed control by the retailer. If the quantities are not automatically defaulted, the user can scan each unit individually to ensure that the correct quantities are recorded. If the quantities are defaulted, only a visual inspection can be done since scanning the item will automatically increase the received quantity.

Direct Delivery UIN Tracking

Before confirming a receipt for a direct delivery on the handheld or the PC, SIM validates to see if any items have a Capture Time of Store Receiving. Serial numbers must be entered or scanned when receiving items that require UINs. If a serial number is not scanned, the quantity on the delivery will not increase.

Note: If applying an ASN, the user is prompted if the defaulted quantity is not equal to the number of UINs scanned to be received.

The Receive All option will not be available if UINs are required for at least one of the items on the DSD.

At this time, the ASN message from the supplier does not include any UIN information.

For AGSN receiving, the user enters the received quantity and damaged quantity. UINs are not scanned during the receiving process. SIM generates UINs and prints a ticket for each damaged and received item. A business process should be put in place to put the AGSN tickets on the correct items.

When the delivery is accessed after is has been received, the user can view the UINs that were created.

Direct Exchange (DEX) and Network Exchange (NEX) Receiving

Direct Exchange (DEX) and Network Exchange (NEX) are uniform communications standards. DEX is the means through which a supplier, using a handheld device, can exchange electronic invoicing information with a store's direct store delivery (DSD) system. NEX differs in its delivery system, using the web as opposed to a hand-held cradle.

SIM is designed to support the integration of a supplier's DEX/NEX information into direct delivery-related screens, thereby simplifying the receiving process. Data is transferred to a store's DSD system using the Electronic Data Interchange (EDI) transaction set 894 (delivery/return base record). With the uploaded data, the store user can view, edit, and confirm the information contained in the file before receiving the direct delivery.

A system option controls whether or not expected quantities are defaulted in for direct store deliveries with Dex/Nex. This allows more detailed control by the retailer. If the quantities are not automatically defaulted, the user can scan each unit individually to ensure that the correct quantities are recorded. If the quantities are defaulted, only a visual inspection can be done since scanning the item will automatically increase the received quantity.

Existing POs versus Quick Order Entry

An existing PO is defined as a PO coming from an external system. SIM can receive against such POs with or without an ASN.

SIM also has the ability to create POs on the fly. These can be based on Dex/Nex transactions or be manually entered based on an invoice from the vendor.

Figure 4–17 Direct Store Delivery (new created) Business Process Flow – PC

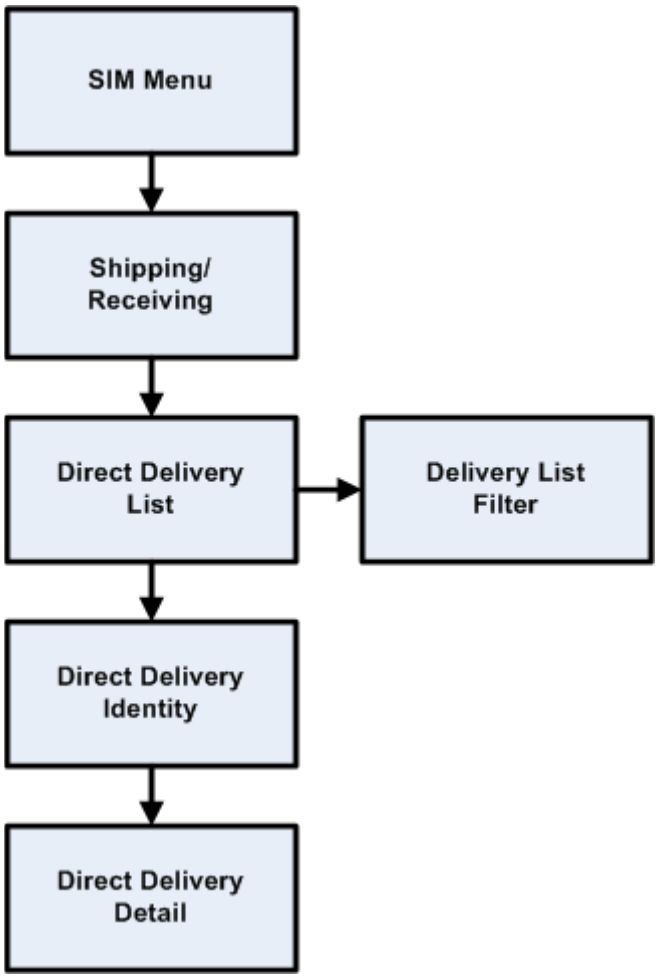


Figure 4–18 DSD Multiple Available ASNs Business Process Flow – PC

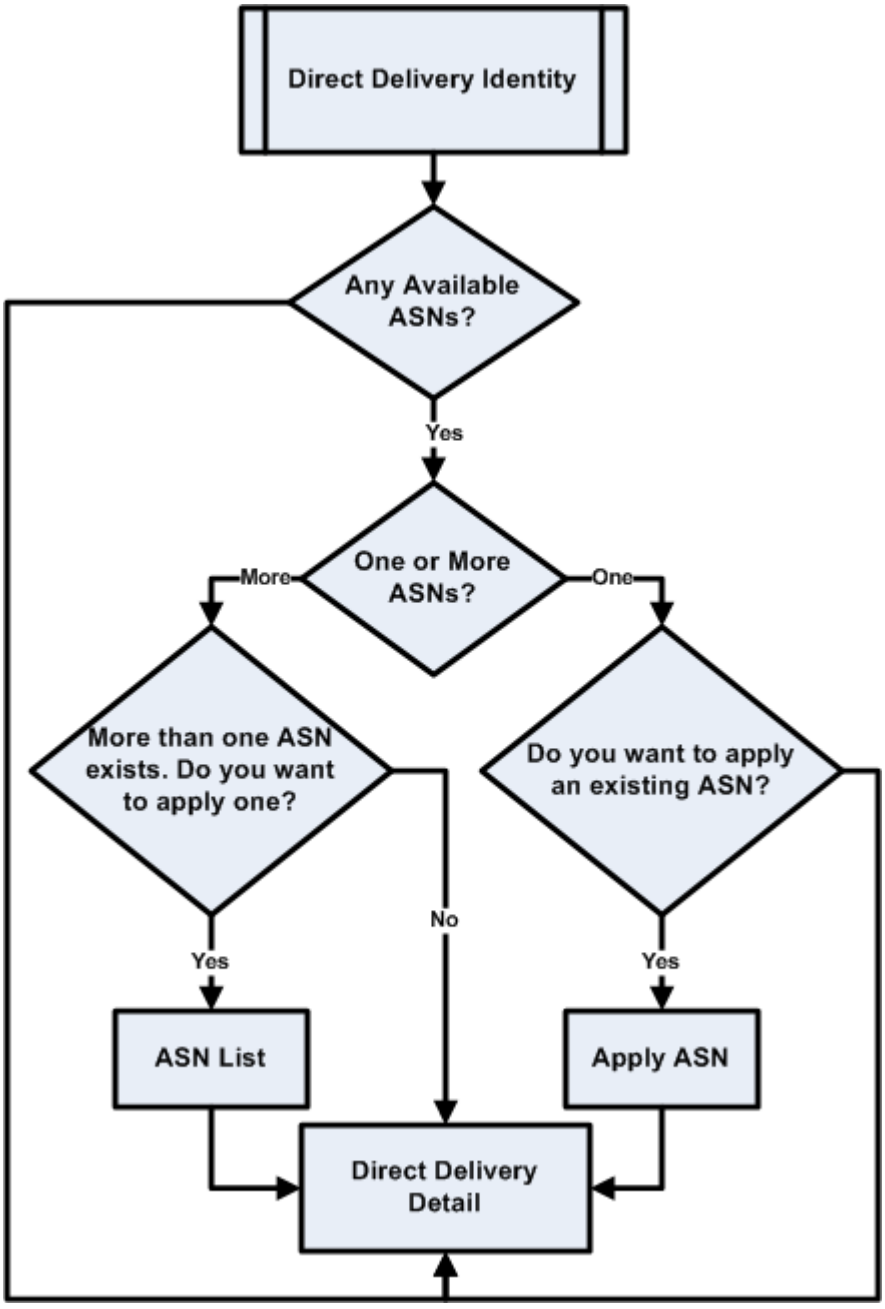
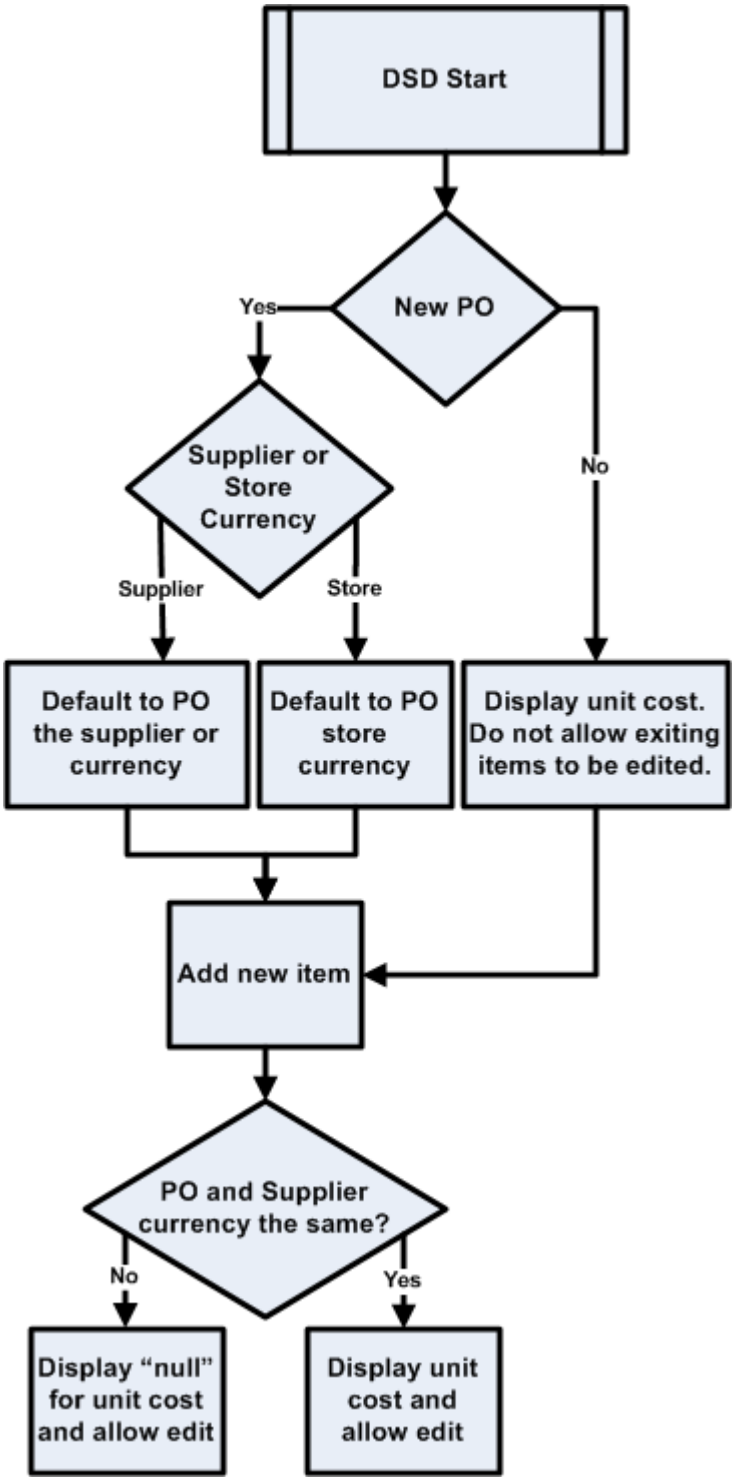


Figure 4–19 DSD Updating and Defaulting Cost Business Process Flow – PC



Receiver Unit Adjustments

During the receiving process, there are situations where it becomes necessary to be able to amend a receipt once it has been completed and sent to the merchandising system for processing. SIM allows users to edit quantities on receipts from a warehouse, direct delivery, and transfer once those shipments have been received.

Unit Receiver Adjustments are available depending on system configurations. There are three separate system configurations corresponding to each receiving function. The configurations represent the number of days a receipt can be adjusted. For example, if the configuration is set to zero, this would disable the unit receiver adjustment functionality. Conversely, if a unit receiver adjustment were set to a value greater than zero, the receipt would be available for the specified amount of days. The users can re-open already received deliveries by clicking **Adjust Delivery**, which is available on the receiving detail screen and then modify the received quantities. Corrected data is then processed and resent to the merchandising system.

Unit Receiver Adjustments are only available on the PC and uses the existing receiving screens.

Receiver Unit Adjustments UIN Tracking

When a receipt is adjusted for an item that requires a UIN, the UIN will need to be added or removed from the transaction. If UINs are added, the received quantity increases by the number of additional UINs added. If UINs are removed from the receipt, the received quantity decreases by the number of UINs removed.

If externally generated receipt adjustments are sent from RMS, an exception is captured. The SOH and receipt will still be updated for the receiver unit adjustment on the backend.

For example:

```
     RMS             SIM
     SOH    Rcpt     SOH    Rcpt
       3       3       3       3

     External receiver unit adjustment done in for -1.
     SOH and receipt updated in both external system and SIM.
     RMS             SIM
     SOH    Rcpt     SOH    Rcpt
RUA   -1      -1      -1      -1
       2       2       2       2
```

A business process should be put in place to resolve the discrepancy on the GUI. See "Resolving UIN Discrepancies" for more details.

When entering the DSD, Warehouse Delivery or Transfer dialogues, the user will be prevented from exiting the RUA transaction until the received quantity equals the number of UINs.

Receipt Adjustments for Transfers should not be allowed for any UIN unless the status of the UIN is in In Stock. The reason for this is that other states such as shipped out, reserved for shipping, unavailable, customer order reserved and so forth have other business transactions against them. The item should be removed from those transactions before the adjustment is allowed. The business workaround would be to add the item back in stock and then remove it through the adjust transfer dialog.

When adjusting receipts for AGSNs, an AGSN would be added or removed from the list. When adding units to the receipt of a Warehouse Delivery, DSD, or Transfer, SIM:

- Creates AGSNs for items that are added or items that increase in quantity
- Prints labels automatically for the newly created AGSNs
- Associates the AGSN to the item without prompting the user to add/scan them manually

In case of a RUA that is reducing inventory, the user will be required to select the AGSN that is being removed from the transaction.

Returns and Return Requests Functional Overview

Returns

SIM allows a store user to look up, create, edit, delete, and complete returns from the store to an external finisher, a company-owned warehouse and/or directly to the vendor. Returns functionality within the SIM system can be accomplished on a PC based deployment, on a wireless handheld device, or on a combination of the two deployment methods.

If the return is to a warehouse (RTW), the user selects the appropriate warehouse from a list. If the return is direct to a vendor (RTV), the user enters the vendor number or uses the search option to identify the vendor for which the items should be shipped.

For the RTV, the user is prompted to enter a reason code for the return if the supplier requires a return authorization code.

The user can also choose to ship the item to an external finisher. Through the context field, the user can specify the purpose of the return. For example, the user can indicate the item is being returned for repair. After the finisher has completed the repair work, a transaction is generated for SIM to receive against. The receipt is done in the warehouse delivery dialogue.

The context field can only be assigned if the return is created in SIM.

Return to warehouse and finisher require the user to select an inventory status at the header level of the return to define whether all items are returned from available or unavailable inventory. For return to supplier, the user may return available and unavailable items on the same return transaction and this would be done by selecting the Use Unavailable flag for the line item. Item quantities can be entered in eaches or cases. Once the applicable quantities are entered, the user is prompted to enter a reason code for the return. The reason codes are retailer defined. Available SOH is decremented for the return except when the item has unavailable inventory. After the reason code is selected, the user may either complete the return or save it to be completed at a later date.

Once a return is dispatched, the available stock on hand is decremented. If the user decided during the return process to source the quantity from unavailable inventory, an additional inventory adjustment is generated with a reason code of returns that moves the stock from unavailable to available.

Once the return is completed, a return document can be printed to be used both as a report and/or a packing slip for the shipment.

Note: It is only possible to return merchandise directly to the vendor (RTV) from SIM if the vendor is allowed to receive returns and if the vendor is allowed to do Direct Store Deliveries.

Returns UIN Tracking

For Return to Warehouse and Return to Vendor transactions, the system will check if the capture time is store receiving for the item. If it is, the user will be required to enter UINs and the quantity field will be equal to the number of UINs entered/scanned for the item. Saving the transaction will move the UINs to Reserved for Shipping status.

If the UIN added to the return is in unavailable status, SIM assumes the user wants to use unavailable inventory for the return. A separate check for unavailable inventory is not needed for UINs.

If the UIN is not assigned to the current store, the UIN is not allowed to be added to the return.

Once dispatched, the UIN will move to Shipped to Warehouse for RTWs, Shipped to Finisher for Return to Finisher, and Shipped to Vendor for RTVs.

An audit record is captured for each status change.

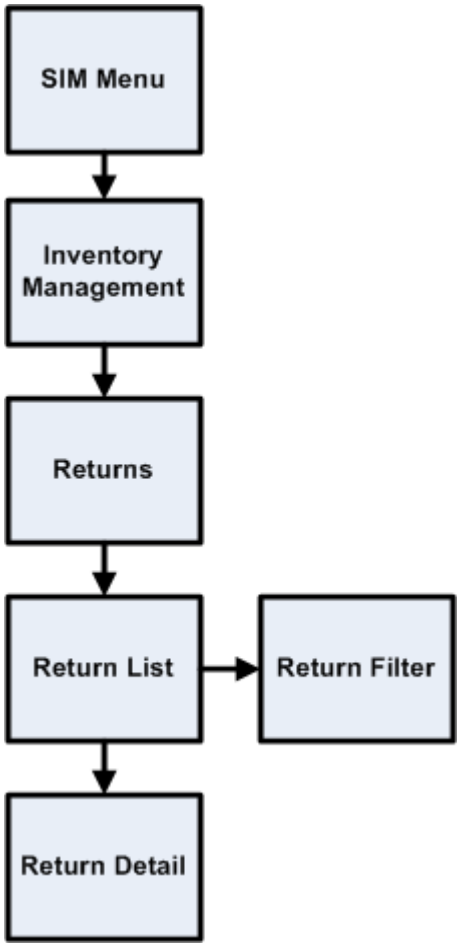
Return Requests

Return requests functionality enables return requests to be fulfilled from a store to a warehouse (RTW) and/or to a vendor (RTV) that were generated using the merchandising system.

A return request might be generated by a safety concern (for example, glass shards are discovered in a product). The functionality can be summed up as a return generated by the merchandising system that can be edited and approved in SIM. Return requests functionality within the SIM system is accomplished only on the PC-based deployment.

Once SIM receives the return request data, it takes over the request and allows store users to add, edit, delete, save, and dispatch the return request. Once the request is deleted or dispatched, a message is sent back to the merchandising system.

Figure 4–20 Return Requests Business Process Flow – PC



Lookups

Item Lookup

SIM provides store users the ability to query basic item information and search for stock in other store locations. All of the lookup functions have filter status on which to search. For example, a user can search for items by item number, description, supplier, and/or merchandise hierarchy. The information that can be searched on and displayed to the user is as follows:

- Search Criteria
  - Item Number (UPC, SKU)
  - Unique Identification Number (UIN)
  - Item Description
  - (Primary) Supplier Name
  - (Primary) Supplier Number
  - Warehouse

  - Finisher
  - Merchandise Hierarchy
  - Ranged indicator
  - User-Defined Attribute (UDA): UDA Text, Value, or Date
- Information Displayed
  - Item Number (SKU)
  - Item Description
  - Item Image
  - Supplier Name
  - Supplier Number
  - Primary UPC/EAN Number
  - VPN
  - Primary Supplier
  - UIN Detail
  - Item/Location ranging
  - Primary Sequence Location
  - Unit Of Measure
  - Simple Pack Conversion
  - Concession/Consignment Item
  - Inventory – defaults to the store the user is logged on to and displays the following categories in units:
    * Total Stock on Hand
    * Available Stock on Hand
    * Shop Floor
    * Backroom
    * Unavailable Stock
    * Transfer Reserved
    * RTV Reserved
    * Ordered Quantity
    * Delivery Bay
    * In Transit
    * Received Today
  - Stock on hand at other store locations (Stock Locator)
  - Customer Orders
  - Department, Class, and Subclass
  - Item differentiators and their related items
  - Primary Pack Size

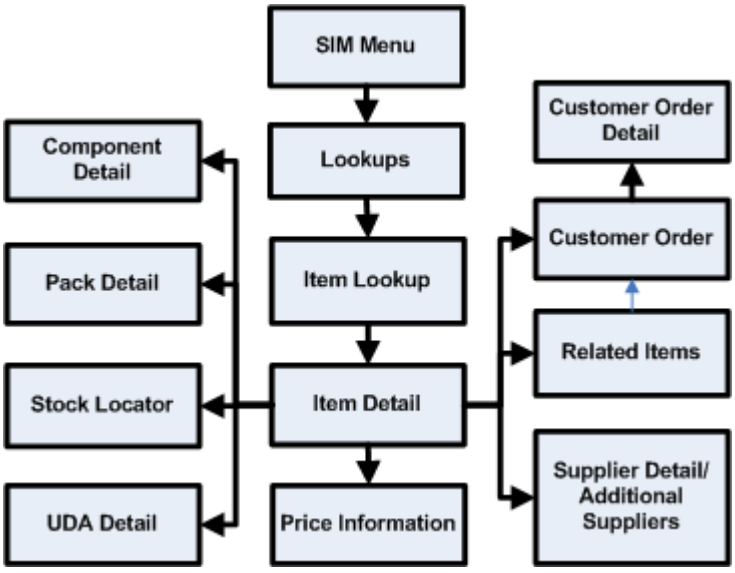
- Pricing – Current and Regular Retail Price
- Price Status – Clearance, Promotional, Market
- Regular Multi Price information
- Price History
- Item Status – active/inactive
- Next allocations – Delivery Date, Warehouse, UOM, Quantity, Timeslots
- Replenishment Method
- Reject Store Orders – for store orders
- Next Delivery Date- for auto replenishment items
- UDA Detail

In an effort to reduce the number of keystrokes, if a lookup on an item is done during the processing of a separate function (that is, a transfer), the ability to use the item directly from the search is enabled. For example, the user can search for an item or supplier directly from the transfer screen, select Use Item, and the information will default into the transfer currently being created.

The handheld will display the same information but also has a walk through lookup function. This feature allows the user to scan a specific item and walk through the different differentiators of the item. The user is then presented with the on hand positions and details of the found item. This is especially useful when a customer cannot find the item, but the store has a very similar item. For example, the user presents a black large T-shirt, but the item the customer actually wants is a red medium sized T-shirt.

The user will also find in this dialog some rudimentary information on customer orders.

Figure 4–21 Item Lookup Business Process Flow – PC



Item Image URLs

SIM enables a customer to define a URL to an image for an item. When the application server is inside a firewall and the referenced images are outside the firewall, the managed server that runs within the application server might need to be started with JVM options defining the proxy server through which URL requests must pass to get outside the firewall, to access images.

The following is an example of these settings:

```
export JAVA_OPTIONS="-Dhttp.proxyHost=myproxy.mycompany.com -Dhttp.proxyPort=80
-Dhttp.nonProxyHosts=localhost|127.0.0.1|*mycompany.com|10.141.*
-Djava.net.preferIPv4Stack=true"
```

These settings must be passed to the managed server JVM during startup. This is usually done within the shell script that launches the managed server.

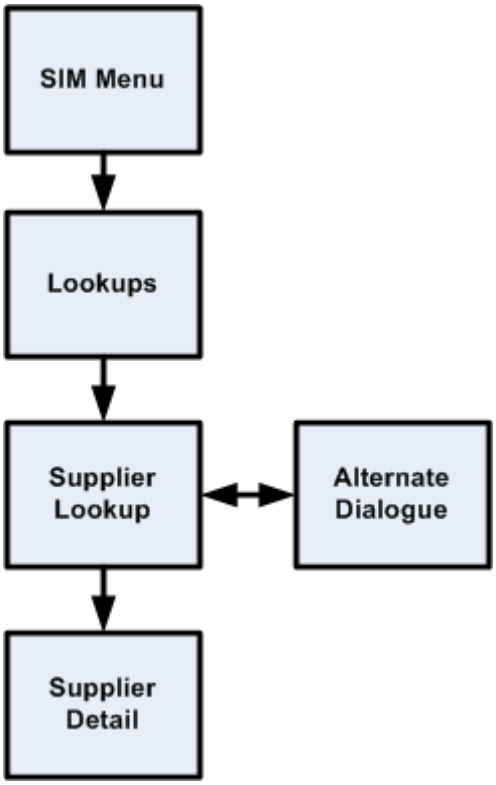
Supplier Lookup

SIM provide users with the ability to query information on suppliers. The following is displayed after a search is performed:

- Supplier Name
- Supplier Number
- Supplier HQ Address, Phone, Fax, Contact, E-mail Address
- Supplier Returns Address, Phone, Fax, Contact, E-mail Address
- Status of the supplier
- Returns Allowed indicator
- Return Authorization Required indicator

As with the item lookup functionality, if a lookup on a supplier is done during the processing of a separate function (that is, a transfer), the ability to use the supplier directly from the search is enabled. This functionality is available on both the handheld and the PC.

Figure 4–22 Supplier Lookup Business Process Flow – PC



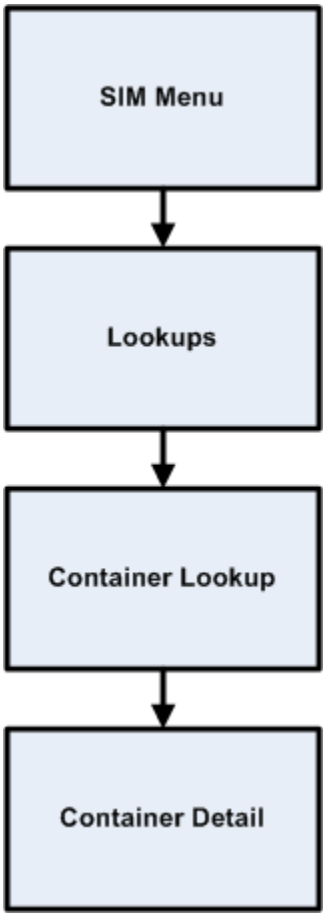
Container Lookup

SIM provides users with the ability to query shipping container information and displays the following:

- Container ID
- ASN Number
- Container Status (Received, In Transit, and so on)
- Item information
- Receipt Date and Time
- From Location
- Number of Cases
- Damages

This functionality is available on both the handheld and the PC.

Figure 4–23 Container Lookup Business Process Flow – PC



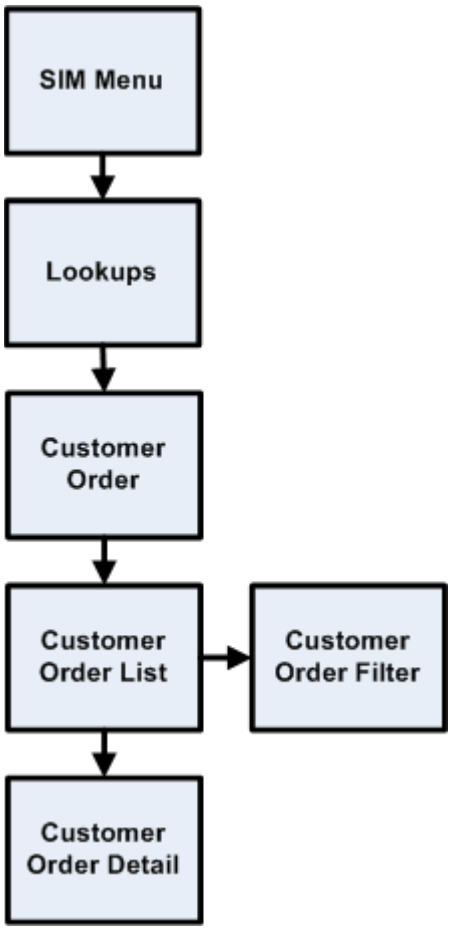
Customer Orders

SIM has the ability to retain some basic information on the customer orders that are responsible for reserving customer specific inventory. The following information can be displayed:

- Item and Item Description
- Customer Order ID
- Reservation type (for example, Layaway, Customer Pickup)
- Status
- Comments (for example, delivery instructions)
- Remaining, fulfilled, reserved, and cancelled quantities
- Last updated

UINs cannot be assigned on Customer Orders, this must be done through an external system. However, the user can view UINs that have been assigned for the Customer Order.

Figure 4–24 Customer Orders Business Process Flow – PC



Unique Identification Number (UINs)

Functional Overview

Retailers who sell items such as electronics, cell phones, weapons, medication, and fresh items often have to track unique numbers or attributes for a single item or a group of items. These numbers are often called serial numbers, batches, unique identification numbers, FCC ID, expiration ID, and so on.

SIM now supports unique identification number logic. The retailer can track the individual instance of an item in SIM from the moment it enters the store until the moment it leaves the store, resulting in better inventory control. UIN tracking is expected to reduce shrinkage, hold stores accountable for individual items, and increase customer satisfaction.

In SIM, UIN functionality allows the user to:

- Lookup a UIN
- View audit trail of UINs
- Resolve UIN discrepancies
- Update UIN status

- Receive UINs (Direct Delivery, Warehouse Delivery, Transfer)
- Count UINs (Stock Counts)
- Perform Inventory Adjustments on UINs
- Prints (Ticketing)

UINs can be captured at the time of sale (Oracle Retail Point-of-Service) or at the time of store receiving (SIM). If the UIN is captured at the time of the sale, Point-of-Service captures the UIN and the UIN is not tracked in SIM. If the UIN is captured at the time of receiving, SIM captures the serial number when it arrives in the store using a direct store delivery or warehouse delivery.

UINs are not allowed for type 2 items, non-inventory items, notional packs, non-sellable simple packs, concession items and consignment items.

Auto Generated Serial Numbers (AGSNs)

SIM also has the ability to auto generate UINs and track the item with that UIN number. The UINs are created during the receiving process and a label is generated for each of these units.

Auto-generated serial numbers will be generated during the receiving process or while performing a stock count or inventory adjustment. If auto generation is being used during the receiving process, the UIN is captured and the UIN information is provided to the user after receipt confirmation.

An auto generation process generates UINs in a sequenced order and assigns to items as needed during DSD receiving, Warehouse Delivery, Transfer Receiving, Stock Counts and Inventory Adjustments.

The default process uses a sequence generated number and is configurable so the customer can enter a desired starting point or hook it into an external service (through customizations).

An audit record is captured for each UIN that has a status updated.

When an item is scanned during the receiving process, the system checks to see if a UIN is required to be captured for the item. If the UIN type is set to Auto Generation (AGSN), the Auto Generation routine will be called and the generated number will be displayed on the UIN popup after the items have been confirmed for the warehouse delivery, DSD, or transfer.

SIM will automatically print an item ticket with the newly generated UIN number.

Note: The print option will only be available for generated UINs. The user will not be able to print UINs that are not auto generated by the system.

- UINs are auto-generated upon receipt confirmation of a warehouse delivery or direct store delivery and labels are printed for each UIN
- UINs are auto-generated for incoming transfers that are received from a store that does not capture UINs
- UINs are auto-generated for inventory adjustments with disposition movement of OUT -> ATS and a ticket prints automatically
- Ability to print/re-print AGSN from Item Ticket Detail screen and UIN Detail screen

- UINs are auto-generated for receiver unit adjustments where the quantity has increased

Note: For externally created adjustments, manual intervention is needed.

AGSN Auto-Ticket Printing

When a new AGSN item is received, SIM automatically prints a ticket.

To improve performance while receiving serialized items, the generated UINs are stored in a print batch table, grouped by a batch ID. This batch ID is pushed to a staging queue. This ends the receiving operation.

A new polling timer is created, which picks the staged UIN print batch message and prints the UINs as a batch through a UIN print batch consumer.

Instead of printing AGSN item tickets directly upon the receipt, the printing now goes through the new polling timer. Printing is done asynchronously, in near real-time, depending on the frequency of the polling timer.

The AGSN report template prints multiple pages (batch UINs) or a single page (single UIN).

UIN AutoNumber

To facilitate the application of serial numbers (UIN), SIM adds a new process that creates the UIN and tracks the item with that number.

During the receiving process, SIM registers how many units are received and generates a label for these units. The process will be identical to how a user receives without capturing UINs, but units are tracked.

The benefits for such a model is the speed of the receiving, which can be done at container level, and removes difficulties some users encounter, for example, trying to find the barcode on large items such as a refrigerator, or determining how to track an item such as a cell phone, which has three barcodes.

For auto-generated serial numbers, SIM bypasses entering individual specific serial numbers at the time of receiving and simply accepts a quantity. This operation is the opposite of normal serial number operations. The quantity entered is then used to generate serial numbers and assign them to the particular item.

AutoGenerateSerialNumberDao contains APIs to retrieve a number of new IDs (or serial numbers) based on the count parameter. This is handled by getting the next values from the AUTO_GENERATE_SN_SEQ sequence. SIM can be modified to generate any sort of serial number the user needs by changing dao.cfg. The user can plug in any class in the `AUTO_GENERATE_SERIAL_NUMBER_DAO=xxx` line and implement any process to generate serial numbers.

If a previously existing auto generated serial number is scanned, it is treated identical to regular serial numbers.

Table 4–5 Enumerations

Enumeration	Description
FunctionalArea	Describes the business functional area and sometimes phase of the business process.
UINType	Describes the type of a UIN. Currently only SERIAL and AGSN are available.
UINStatus	Describes the status of the UIN.
UINCaptureTime	For the specific item and store, it defines the time when a new UIN may be captured and inserted into the data store. Either SALE or STORE_ RECEIVING.
UINAvailability	Used as a parameter when searching for records based on availability.
UINActionType	This represents action type that triggered a UIN update (namely from a web service.).

Table 4–6 UINStatus

Name	Code	Description	Comment
IN_STOCK	0	In Stock	This status can be sold, inventory adjusted, stock counted, shipped, reserved for shipping and reserved for sale.
SOLD	1	Sold	This status is considered a final status and can only be changed through a stock count, return or inventory adjustment.
SHIPPED_TO_ WAREHOUSE	2	Shipped To Warehouse	This status is considered a final status and can only be changed through another receipt or special inventory adjustment.
SHIPPED_TO_STORE	3	Shipped To Store	This status can be changed to in stock when the item is received, unavailable if the item is damaged during return or removed from inventory in case it is short received.
RESERVED_FOR_ SHIPPING	4	Reserved For Shipping	This status indicates UIN on return or transfer.
SHIPPED_TO_ VENDER	5	Ship To Vendor	This status is considered a final status and can only be changed through another receipt or special inventory adjustment.
REMOVED_TO_ INVENTORY	6	Remove From Inventory	Set when an item is removed from stock. Only can be changed if item is moved back into inventory.
UNAVAILABLE	7	Unavailable	Set when inventory adjustment is made to unavailable, damaged received quantities or when item is reserved for customer orders.
MISSING	8	Missing	Will be set when a stock count can not find the serial number or the item goes missing during a shipment.
IN_RECEIVING	9	In Receiving	This means that a receipt is In Process but has not yet been confirmed. This can occur during DSD Receiving, Warehouse Receive or Transfer Receiving.
CUSTOMER_ RESERVED	10	Customer Reserved	This status will be set when Oracle Retail Point-of-Service uses the customer order Web service to communicate a UIN that is reserved for a customer order.

Table 4–6 (Cont.) UINStatus

Name	Code	Description	Comment
CUSTOMER_ FULFILLED	11	Customer Fulfilled	This status will be set when Oracle Retail Point-of-Service uses the customer order web service to communicate a UIN that is fulfilled for a customer order.
SHIPPED_TO_ FINISHER	12	Shipped To Finisher	The state which an item should be in when receiving from a Finisher.
UNCONFIRMED	99	Unconfirmed	This means a UIN has been scanned or entered but has not yet been processed. The UIN is in a temporary state and can move from None to any other state during validation. Note: The functional identifier will not exist until the transaction has been completed.

Note:

- UIN **Open** Status = IN_STOCK, RESERVED_FOR_SHIPPING, UNAVAILABLE, CUSTOMER_RESERVED and IN_RECEIVING.
- UIN **Closed** Status = SOLD, MISSING, SHIPPED_TO_STORE, SHIPPED_TO_WAREHOUSE, SHIPPED_TO_VENDOR, SHIPPED_TO_FINISHER, REMOVE_FROM_INVENTORY and CUSTOMER_FULFILLED.

Table 4–7 UINAvailability

Name	Description
OPEN	Open
CLOSED	Closed
ALL	All

Used as a parameter when searching for records based on availability.

Table 4–8 UINActionType

Name
SALE
RETURN
VOID_SALE
VOID_RETURN

This represents action type that triggered a UIN update (namely from a web service).

Auditing

Any time a status change occurs for a UIN, an audit record is captured and is available for viewing on the UIN History screen.

UIN Setup

A store parameter allows the user to turn on/off UIN functionality by store. Multiple system parameters control the purging of UIN information.

SIM provides a store/class level setup for UIN attributes. These attributes can be auto-defaulted in based on a system parameter. When attributes are added or modified at the class level, the attributes will be applied to each item/location level for all items that belong to the specified department/class.

The UIN attributes screen is required for standalone implementations of SIM. The UIN Attributes screen should not be used if the retailer plans to pull attributes from an external system.

UIN attributes include the following:

- Type of UIN (AGSN/Serial Number)
- Capture Time (Store Receiving/Sale)
- UIN Label
- Ticket Format (AGSNs)
- External system create UIN

UIN Status

Each time an item with a UIN is scanned, SIM captures the status of that item. Depending on the functional area for which that item is scanned, a different status will be assigned. This feature allows SIM to ensure data integrity and provide an audit trail of the life of the item.

Before any transaction is completed (dispatched or confirmed), SIM validates that the status of the items on the transaction are still valid.

For example, a UIN on a transfer might be invalid if a stock count cannot find the item and move the **Reserved for Shipping** status to **missing**. The item will stay on the transaction, but the user must remove it before dispatching. SIM lets the user know the item is not in a valid status anymore.

UIN Statuses

Unconfirmed

A UIN has been scanned or entered but has not yet been processed. The UIN is in a temporary state and can move from Unconfirmed to any other state during validation.

In Stock

The item is in stock and can be sold. This status is usually achieved after an item is received, returned or when it is fixed from a repair.

In Receiving

A receipt is In Process but has not yet been confirmed. This can occur during DSD Receiving, Warehouse Receive or Transfer Receiving.

Sold

The item has been sold to a customer. The UIN status can get set to Sold through the new Real Time Point-of-Service Web service.

Reserved For Shipping
Any time a transfer or a return is created and saved the UIN is marked as Reserved For Shipping. Only UINs in "In Stock" and "Unavailable status" will be allowed to be shipped.

Shipped To Store
When a store-to-store transfer is dispatched, the status of the UIN is set to Shipped To Store. In order to update the UIN to Shipped to Store a UIN must be In Stock or Reserved for Shipping.

Shipped To Warehouse
When a warehouse return is dispatched, the status of the UIN is set to Shipped to Warehouse. In order to update the UIN to Shipped to Warehouse a UIN must be In Stock, Reserved for Shipping, or Unavailable status.

Shipped To Vendor
When a vendor return is dispatched, the status of the UIN is set to Shipped to Vendor. In order to update the UIN to Shipped to Vendor a UIN must be In Stock, Reserved for Shipping, or Unavailable status.

Shipped To Finisher
When a finisher return is dispatched, the status of the UIN is set to Shipped to Finisher. In order to update the UIN to Shipped to Finisher a UIN must be In Stock, Reserved for Shipping, or Unavailable status.

Removed From Inventory
A UIN will be updated to Removed From Inventory using either an Inventory Adjustment or a Short Receipt. In order to update the UIN to Removed From Inventory a UIN must be In Stock, Shipped to Store or Unavailable status.

Missing
A UIN will be updated to Missing when performing a stock count. If an item is not found that is currently In Stock, Reserved for Shipping or Unavailable, the UIN will be updated to Missing.

Unavailable
A UIN will be updated to Unavailable either through an Inventory Adjustment or a Damaged Receipt. A UIN must be in either In Stock, Sold, Shipped to Store, Customer Fulfilled, Removed from Inventory, or Missing Status before it can be moved to this status.

Customer Reserved
This status will be set when the Oracle Retail Point-of-Service uses the customer order Web service to communicate a UIN that is reserved for a customer order:

- The selling service to validate the item is valid to be sold will be used to validate that the UIN is available to be reserved.
- A UIN must be in either In Stock, Customer Order Fulfilled before it can be moved to this status.

Customer Fulfilled

This status will be set when Oracle Retail Point-of-Service uses the customer order web service to communicate a UIN that is fulfilled for a customer order:

- The selling service to validate the item is valid to be sold will be used to validate that the UIN is available to be fulfilled.
- A UIN must be in either In Stock or Customer Order Reserved before it can be moved to this status.

Resolving UIN Discrepancies

UINs can be resolved in multiple ways, depending on what the discrepancy is.

The user can view discrepancies on the Resolution List screen. The UIN Resolution screen will display all exception records that were created due to attempting a status change that is not allowed using one of the following:

- UIN Update Status Web Service
- Customer Order Web Service
- Externally generated Receipt Adjustments

When a UIN store mismatch occurs, an e-mail notification is sent to the store with which the UIN was originally associated. This applies to Transfers, Inventory Adjustments and Stock Counts. These discrepancies do not appear on the Resolution list screen, instead the notification will occur through an e-mail and can be resolved by adding the item to a Problem Line stock count or resolving it through an inventory adjustment.

Resolving the UIN record on the UIN Resolution screen does not resolve the discrepancy on the transaction. The recommendation is to resolve the discrepancy by fixing the issue on the transaction or by doing an inventory adjustment:

- The UIN discrepancy can be resolved directly through the transaction from where the discrepancy originated. This is the recommended business process.
- The user can check the **UIN Discrepancies** flag when creating a Problem Line count using Product Group setup. This will add discrepant UINs to the count and resolve the discrepancy through completion of the stock count.
- The status of the UIN can be updated directly from the UIN resolution screen. This automatically marks the record as **resolved**. This does not resolve the inventory discrepancy.
- The UIN record can be moved to **resolved** on the Resolution screen by clicking **Resolve** from the UIN (without updating the status or the inventory).

For Third Party stock counts, UIN discrepancies can be resolved through the Rejected Items screen. If the UIN is not present for an item that requires a UIN when the third party count is uploaded to SIM, the record will be written to the Rejected Items table for later resolution. The Rejected Items screen allows the user to assign a serial number for those items.

Whenever an invalid UIN is scanned on any type of stock count, the invalid UIN will appear on the Rejected Items screen. A user with the proper permissions can review the invalid scans and assign a valid UIN to bring the UIN back into the stock count if desired.

Examples of Resolving Discrepancies

Example 1 – Store Mismatch: Allow Unexpected UINs parameter is set to Yes

1. Transfer sent from Store A to Store B.
2. Store B receives the transfer. Item 1 was not on the transaction, however it did get shipped on the truck so Store B receives the unexpected UIN.
3. The UIN is now associated with Store B and an e-mail will be sent to Store A to notify them of the discrepancy. The item/UIN was not on the transaction and therefore the item/UIN is still reflected in Store A's inventory.
4. The user can create a problem line stock count at Store A and check the UIN Discrepancies flag. This action will place Item 1 on the stock count and resolve the inventory discrepancy once the count is completed. The discrepancy could also instead be resolved using an inventory adjustment to move the UIN out-of-stock.

Example 2 – Store Mismatch: Allow Unexpected UINs parameter is set to No

1. Transfer sent from Store A to Store B.
2. Store B receives the transfer. Item 1 was not on the transaction, however it did get shipped on the truck so Store B attempts to receive the unexpected UIN but SIM does not allow it.
3. The UIN is still associated with Store A. Store B will have to call Store A and have them create a new transfer so Item 1 moves out of Store A and into Store B. An exception record is not created on the Resolution List screen for either store since SIM never allowed the UIN to change status from one store to another.

> **Note:** If the user chooses to update the status using the Resolution list screen, they still must create the transfer so that the inventory gets updated correctly.

4. Once the status has been updated by Store A, the Store B user can now receive the item.

Example 3 – Resolution List Screen RUA

Updating UIN Status at store you are logged in to.

1. RUA is done in RMS for Direct Delivery at Store A.
2. Exception record created for Store A due to the external adjustment and appears on Resolution List screen in Store A.
3. The user logs into store A and goes to Resolution List screen to find exception record.
4. The user should go to the Direct Delivery and click **Adjust Delivery** and manually add or remove UINs as necessary. Adjusting the delivery will update the UIN to the correct status.

5. The user can click **Resolve** from UIN Resolution screen to indicate they have manually resolved the discrepancy. The **Update UIN Status** button is used as an additional check and allows user to update status in case it is ever needed. It is a place where the user can update the status or view an audit trail of the UIN. Ideally, the user will be adjusting the transaction itself which should update the UIN status to where it needs to be. For example, removing UIN from DSD moves UIN status to Removed from Inventory. Adding a UIN to DSD moves UIN status to In Stock.

Example 4 – Resolution List Screen: Customer Order Web service

1. Customer Order Web service calls SIM to move item to Customer Order Reserved and the item/UIN is not in stock.
2. SIM records a discrepancy error on Resolution List screen.
3. User creates an inventory adjustment to bring the UIN In Stock.
4. User can log in to the store called by the web service and update the status to Customer Order Reserved using Resolution List screen.

Example 5 – Update UIN Status Web Service Processing ACTION = SALE or VOID-RETURN

1. UIN is found and is in one of the following statuses:
   - Unavailable
   - Sold
   - Reserved for Shipping
   - Shipped to Store
   - Shipped to Warehouse
   - Shipped to Vendor
   - Missing
   - Customer Order Reserved
   - In Receiving
   - Removed from inventory
2. UIN cannot be updated to Sold if it is in one of the previous statuses, so an exception record is created and appears on the Resolution List screen for the store.
3. Depending on the integration, the user must update the status of the UIN manually from the UIN Resolution List screen and re-process the transaction, or just update the status of the UIN.

   The unified web service will not update the SOH if the status does not match. The UIN status update web service, on the other hand, will fail independently from the sales transaction web service or ReSA upload file. Fixing the problem for the UIN status update web service will only require a status update. The unified web service call might require a status update to In Stock and a re-process of the entire record.

   Note: User might need to remove the UIN from the physical transaction. Depends on what status it was in. For example, if it was in Shipped to Store, the user should, from a business perspective, go to the transfer and remove the item/UIN from the transfer.

Example 6 – Update UIN Status Web Service Processing ACTION = RETURN or VOID-SALE

1. UIN is found and is in a state other than Sold.
2. Status cannot be updated since it is not in sold status and exception is created and appears on Resolution List screen.
3. Access exception record from Resolution List screen and update the status.

5

System and Store Administration

Under the administration section, the user can find all system setup tasks and often corporate executed tasks:

- Product Group and Product Group Scheduler

  This feature allows customers to set up recurring events with sets of items.

- Ad Hoc Stock Counts

  This section allows the user to set up ad hoc stock count variance levels for each merchandise hierarchy at the class level.

- UIN Resolution

  The UIN Resolution screen allows the user to view and resolve UIN status discrepancies. The user can also view an audit trail of the discrepant item's UIN history.

  Note: Resolving discrepancies from the UIN Resolution list screen does not resolve the inventory discrepancy. See Resolving Discrepancies section for more detail.

- Security

  The user can create and modify roles. When creating a new role, it is possible to add a variety of privileges for the handheld or PC. In addition to general restrictions for functional areas, the user is also able to secure the different types of product groups that exist and the reason codes a user has access to for creating inventory adjustments.

- Technical Maintenance

  Several technical functions can be controlled under this header:

  – UI Configuration

    This feature allows the user to configure font type and size, color scheme and icons by theme. In addition translated values can be modified through this dialog as well.

  – Polling Timers

    Polling timers allow the user to identify how often the system will check the SIM integration layer for new messages generated by external systems.

- Staged Messages

  The staged messages UI allows a user to validate the content of message and if needed manually restarts the message for polling purposes.

- Setup
  - SIM Stores

    Management of SIM managed stores, setup of buddy stores, and auto receiving for store transfers.

    * SIM Managed Stores

      User can setup those stores that will use SIM. This prevents the store from publishing RIB Messages to the external system when auto-receiving.

    * Buddy Stores

      SIM allows for the concept of Buddy Stores. Buddy Stores can be set up to indicate groups of stores that can transfer merchandise from one store to another. The concept does enforce transfer zones if used in the Oracle Retail Merchandising System.

    * Auto Receive Stores

      SIM allows users to set up Stores at which transfers are automatically received when shipped.

  - Administration Parameters

    SIM has many application parameters that allow clients to customize the product according to their business. The application parameters are split into system and store options. System option parameters allow a user to change the parameter for the entire system and all stores. Store option parameters are only specific to the store the current user is logged in to.

  - Formats and Printer Selection

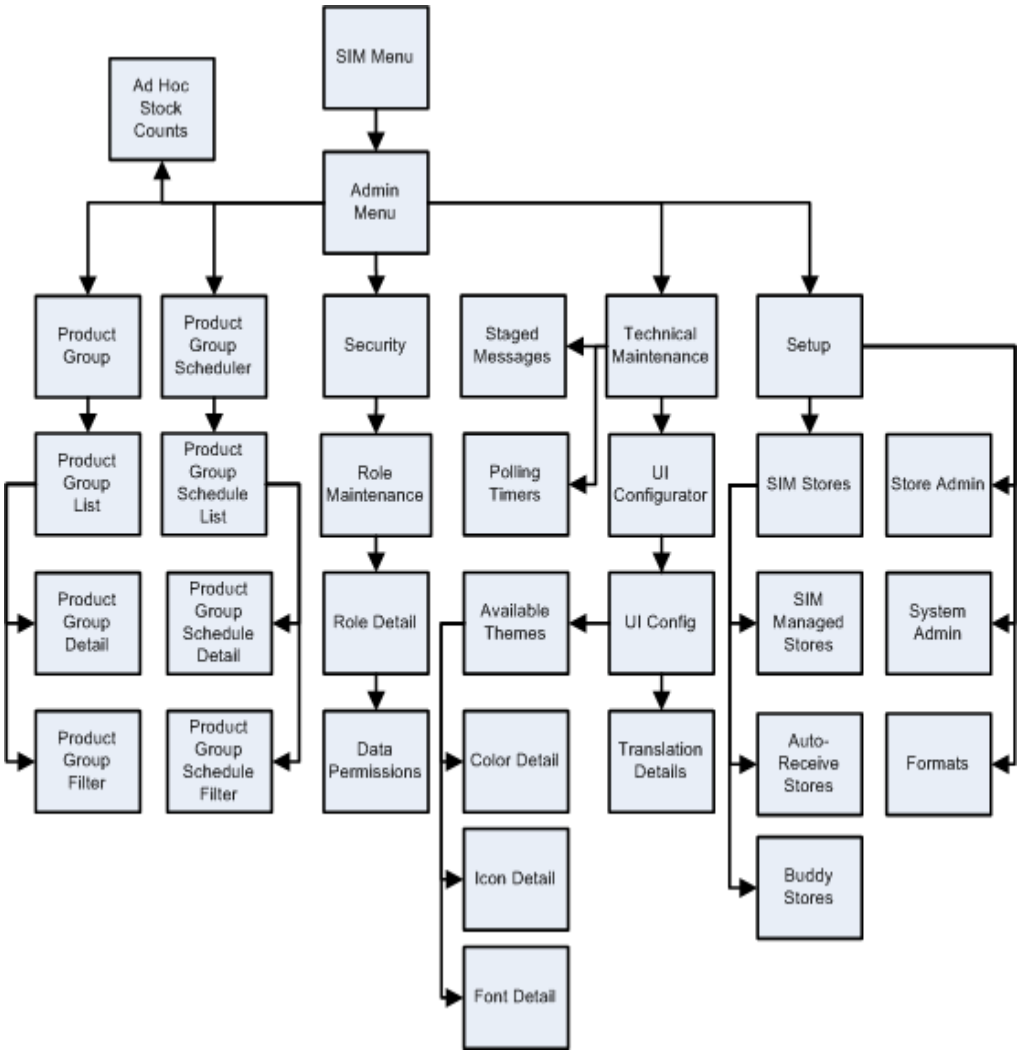
    Tickets and labels can be setup here and a default printer can be assigned to them. In addition, it is possible to assign a default printer to print reports.

    Note: Tickets and labels need to be created in the printing tool used to print them. These screens are just for printer and type setup.

  - UIN Attributes
  - Inventory Adjustment Reason

    Inventory adjustment reason codes help control the loss or unexpected gain of items for the general ledger and stock ledger. SIM has the ability for the user to set these up to match the external merchandising system. It is also possible to hide or modify the disposition of the existing reason codes.

Figure 5–1 Store Administration



Product Groups/Scheduler

Within the System Administration screens is the ability for a store user, with the proper security, to create any number of groups of items to be used within the SIM application. These groups can be comprised of entire areas of the merchandise hierarchy (for example, an entire subclass) or can be simply a group of individual and unrelated items. Depending on the product group, the user can setup additional details such as:

- Tolerances
- Counting Method (Guided/Unguided/Third Party)
- Hierarchy Breakdown
- Recounts
- Item Status
- Stock on Hand
- Expiration

- Delivery Dates
- Auto Authorize
- Problem Line Parameters

Product groups can be created for:

- Unit Stock Counts
- Unit and Amount Stock Counts
- Problem Line Stock Counts
- Pick Lists (Shelf Replenishment)
- Wastage
- Item Requests

Once the groups are created, the user has the ability to schedule how often each group is to be counted or ordered. Using a calendar wizard, the user selects the count group and whether it is to be counted daily, weekly, monthly, or yearly. One or more stores can be assigned to the schedule, depending on stores the user has access to. SIM maintains these schedules and automatically prompts users to complete the counts at their scheduled times. Product group schedules can be used for:

- Unit Stock Counts
- Unit and Amount Stock Counts
- Problem Line Stock Counts
- Wastage
- Item Requests

Store Administration

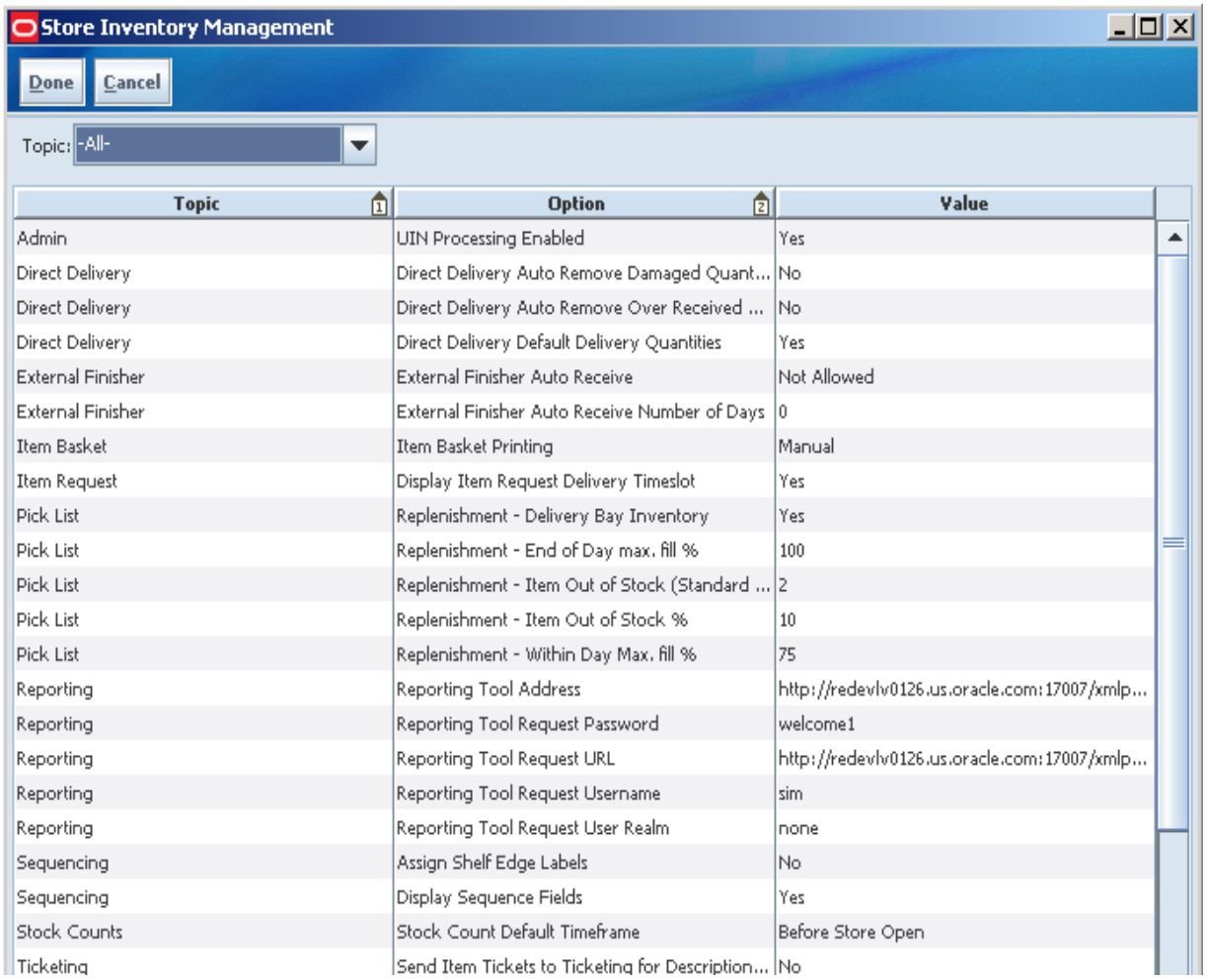
The store administration functionality allows you to set values for options that control a variety of system behaviors. The values of these system options apply only to your location.

Set Store Options

Go to **Main Menu –> Admin –> Setup –> Store Admin**.

The Store Admin window opens.

Figure 5–2 The Store Admin Window



1. Select the option that you want to modify.
2. Double-click the Value field and set the option value in either of these ways:
   - Select a value from the list.
   - Enter an appropriate value in the field.
3. Click **Done**. You return to the Setup menu. Click **Done** again to return to the Admin menu.

Store Administration Options Table

The following table lists the store administration options in alphabetical order and describes each option.

Table 5–1 Store Administration Options

Topic	Store Administration Option	Valid Values	Default Value	Description
Admin	Enable Item Disposition in Transaction Updates	Yes, No	Yes	If Yes, the system reads the reason code in the web service message and performs the inventory adjustment. If the reason code is not present, the system processes the return/void and increments the SOH. If No, the system does not look for the reason code in the web service message.
Admin	UIN Processing Enabled	Yes, No	No	This parameter dictate whether any of the UIN functionality is available within SIM for each store. If the parameter is **No**, then none of the UIN buttons or fields will be present in the application.
Direct Delivery	Direct Delivery Auto Remove Damaged Quantity	Yes, No	No	**Yes**: When confirming the transaction, all damaged items are removed and put on the audit trail. **No**: Any damaged quantities recorded for the delivery stay on the transaction.
Direct Delivery	Direct Delivery Auto Remove Over Received Quantity	Yes, No	No	**Yes**: The user is allowed to add any quantity for the DSD ASN, but any quantity greater than the expected quantity is removed from the transaction. After the user confirms the transaction, the user is prompted that any over-received quantities are removed. **No**: Follows standard DSD receiving process.
Direct Delivery	Direct Delivery Default Delivery Quantities	Yes, No	No	**Yes**: SIM defaults the quantities from the Dex/Nex or ASN. **No**: Does not default the quantities. The user can use the scan-scan feature.

Table 5–1 (Cont.) Store Administration Options

Topic	Store Administration Option	Valid Values	Default Value	Description
External Finisher	External Finisher Auto Receive	Not allowed, External Message, Date Driven	Not allowed	This parameter drives the following functionality: ■ **Not allowed** follows standard receiving process. ■ **External message** receives the full finisher delivery the moment an ASN transaction arrives which indicates that the delivery needs to be auto-received. ■ **Date Driven** looks at a secondary store option (External Finisher Auto Receive number of Days) to determine how many days the transaction stays open before the transaction is fully received. If the parameter is set to **0**, the transaction auto-receives on the ETA date.
External Finisher	External Finisher Auto Receive Number of Days	0-99 ■ **0** means immediate receiving ■ **1** means today (EOD) ■ **2** means EOD tomorrow ■ **x** means EOD *x* days starting from today	0	SIM auto-receives any external finisher delivery ASN that has not been closed *x* days after the ETA date or the create date, depending on whether the ETA date is set or whether the auto-receive External Finisher delivery parameter is set.
Item Basket	Item Basket Printing	Manual, Automatic	Manual	If the value of this option is **Automatic**, Item Basket ticket is printed when the transaction is complete. If the value is **Manual**, SIM does not automatically print the ticket.
Item Request	Display Item Request Delivery Timeslot	Yes, No	No	If the value of this option is **Yes**, SIM displays the timeslot information in Item Request and Item Lookup.
Pick List	Replenishment – Delivery Bay Inventory	Yes, No	Yes	This option allows you to turn on or off the delivery bay functionality.
Pick List	Replenishment – End of Day max. fill %	Numeric 0 to 100	100	This configurable percentage allows each store to set its own fill percentage for creating end-of-day pick lists.
Pick List	Replenishment – Item Out of Stock (Standard UOM)	Numeric	2 (standard unit of measure)	Use this option to set a variance for out-of-stock items on the shelf. The out-of-stock field is used when receiving warehouse deliveries. If the quantity on the shelf is less than the amount in this field, the item appears as an out-of-stock item.

Table 5–1 (Cont.) Store Administration Options

Topic	Store Administration Option	Valid Values	Default Value	Description
Pick List	Replenishment – Item Out of Stock %	Numeric	10	Stores can set a variance for out-of-stock items on the shelf. The out-of-stock field is used when receiving warehouse deliveries. If the percentage on the shelf (shopfloor divided by capacity) is less than the percentage specified by this option, the item appears as an out-of-stock item.
Pick List	Replenishment – Within Day Max. fill %	Numeric 0 to 100	75	This configurable percentage allows each store to set its own fill percentage for creating within-day pick lists.
Reporting	Reporting Tool Address	Text	(None)	This option can specify the URL of the reporting tool. This address is used to start the reporting tool when the user clicks the **Reports** button on the Main Menu. This address is used to display all operational reports, and user intervention is required to print reports.
Reporting	Reporting Tool Request Password	Text	(None)	This option can specify the password of the specific report request.
Reporting	Reporting Tool Request URL	Text	(None)	This option can specify the URL of the specific report request.
Reporting	Reporting Tool Request Username	Text	admin	This option can specify the username of the specific report request.
Reporting	Reporting Tool Request User Realm	Text	None	This option should be used if SSO policy includes a user realm (known as Company Name in SSO terminology); if not, the property should be left as **none**. <*OSSO_USER*>, <*OSSO_PASSWORD*>, and <*OSSO_USER_REALM*> should be used if the Oracle BI Publisher instance does not have local Oracle BI Publisher users and uses Oracle Single Sign-on instead for security.
Sequencing	Assign Shelf Edge Labels	Yes, No	No	If the value of this option is **Yes**, users are required to assign shelf edge labels for sequencing.
Sequencing	Display Sequence Fields	Yes, No	No	Indicates whether or not sequencing fields will be displayed in the Item Lookup screen.
Stock Counts	Stock Count Default Timeframe	Before Store Open, After Store Close	Before Store Open	The setting of this option determines when the stock count is performed in relation to store opening hours for daily sales processing. It defines the default value for the Stock Count screen.
Ticketing	Send Item Tickets to Ticketing for Description Change	Yes, No	No	If the value of this option is **Yes**, a ticket is generated to be printed when the description of an item changes.

Table 5–1 (Cont.) Store Administration Options

Topic	Store Administration Option	Valid Values	Default Value	Description
Ticketing	Send Item Tickets to Ticketing for Price Change	Yes, No	Yes	If the value of this option is **Yes**, when a new approved price change enters SIM, SIM automatically creates an item ticket record that is displayed on the Item Ticket List screen.
Ticketing	Send Item Tickets to Ticketing for QR Code Change	Yes, No	No	Indicates SIM generates a new ticket if the QR code changes based on the optional time/date. If no time/date is defined, it is generated immediately; if a time/date exists, it will be generated by a batch program.
Ticketing	Send Shelf Edge Labels to Ticketing for Description Change	Yes, No	No	If the value of this option is **Yes**, a label is generated to be printed when the description of an item changes.
Ticketing	Send Shelf Edge Labels to Ticketing for Price Change	Yes, No	Yes	If the setting of this option is **Yes**, when a new approved price change enters SIM, SIM automatically creates a shelf edge label record that is displayed on the Item Ticket List screen.
Ticketing	Send Shelf Edge Labels to Ticketing for QR Code Change	Yes, No	No	Indicates SIM generates a new label if the QR code changes based on the optional time/date. If no time/date is defined, it is generated immediately; if a time/date exists, it will be generated by a batch program.
Transfers	Store Auto Receive	Not Allowed, External Message, Date Driven	Not Allowed	This parameter drives the following functionality: ■ **Not allowed** follows standard Transfer receive process. ■ **External message** receives the full store delivery the moment a transaction arrives which indicates that the delivery needs to be auto-received. This feature works in correlation with the store auto-receive screen information configured under Admin > SIM Store > Auto-Receive Stores. ■ **Date Driven** uses a secondary store option (Store Auto Receive Number of Days) to determine how many days the transaction stays open before the transaction is fully received.

Table 5–1 (Cont.) Store Administration Options

Topic	Store Administration Option	Valid Values	Default Value	Description
Transfers	Store Auto Receive Number of Days	0-99 ■ **0** means immediate receiving ■ **1** means today (EOD) ■ **2** means EOD tomorrow ■ **x** means EOD *x* days starting from today	0	A batch program auto-receives any store transfers that have not been closed *x* days after they have been shipped, if the store auto-receive parameters are set.
Warehouse Delivery	Warehouse Auto Receive	Not Allowed, External Message, Date Driven	Not Allowed	This parameter drives the following functionality: ■ **Not allowed** follows standard Warehouse receiving process. ■ **External message** receives the full warehouse delivery the moment an ASN transaction arrives. The external message indicates that the delivery needs to be auto-received. ■ **Date Driven** looks at a secondary store option (Warehouse Auto Receive number of Days) to determine how many days the transaction stays open before the transaction is fully received. If the parameter is set to **0**, the transaction will auto-receive on the ETA date.
Warehouse Delivery	Warehouse Auto Receive Number of Days	0-99 ■ **0** means immediate receiving ■ **1** means today (EOD) ■ **2** means EOD tomorrow ■ **x** means EOD *x* days starting from today	0	A batch program auto-receives any warehouse deliveries that have not been closed *x* days after the ETA date or the create date, depending on whether the ETA date is set and whether the auto-receive warehouse delivery parameter is set.

System Administration

The system administration functionality allows you to set values for options that control a variety of system behaviors. The values of these system options are applied to all locations.

The system options are in these general categories:

- Receiving and shipping options
- Audit options

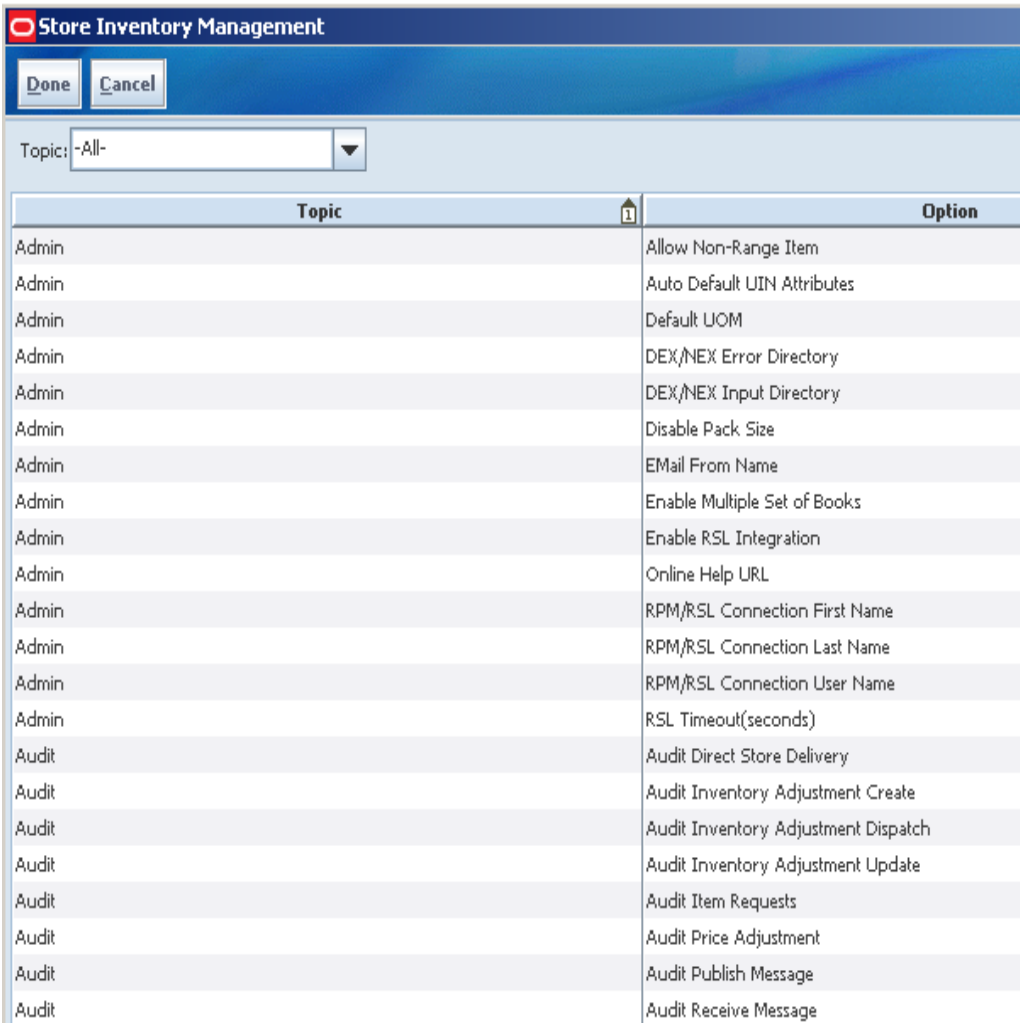
- Transaction adjustment options
- Days-to-hold options
- System usability options
- E-mail options
- Stock count options
- Warehouse receiving options
- Transfer options
- Time Zone options
- Miscellaneous options

Set System Options

Go to **Main Menu –> Admin –> Setup –> System Admin**.

The System Admin window opens.

Figure 5–3 The System Admin Window



1. Select the option that you want to modify.
2. Double-click the Value field and set the option value in either of these ways:
   - Select a value from the list.
   - Enter an appropriate value in the field.
3. Click **Done**. You return to the Setup menu. Click **Done** again to return to the Admin menu.

System Administration Options Tables

The following tables list the system administration options in each general category and describe each option.

Table 5–2 Topic: Admin Options

System Administration Option	Valid Values	Default Value	Description
Allow Non-Range Item	Yes, No	Yes	This option gives stores the ability to add non-ranged items to functional areas in the application.
Auto Default UIN Attributes	Yes/No	No	Dictates whether SIM auto-defaults in the UIN attributes. If the parameter is **Yes**, then when a new item is created using the RIB, SIM auto-defaults UIN attributes that were set up at the department class from the SIM UIN Attributes screen. If the parameter is **No**, then the UIN attributes are not defaulted in. Instead, the UIN attributes are defaulted from the external system. The parameter should be set to **No** if an external system controls the setup of item UIN attributes. In this case, SIM must not auto-default the UIN values as this action can override the intent of the external system.
Default UOM	Cases, Standard UOM	Cases	This option allows the store to select the default unit of measure that the store will normally use. This system option is ignored in the stock counts functional area.
DexNex Error Directory	Editable text field	na	Dictates where the error directory is located for Direct Delivery DexNex files. For example: `/u00/webadmin/product/10.1.3_9/OAS/user_projects/domains/java_domain/error` See the *Oracle Retail Store Inventory Management Operations Guide* for details on DexNex batch file processing.
DexNex Input Directory	Editable text field	na	Dictates where the input directory is located for Direct Delivery DexNex files. For example: `/u00/webadmin/product/10.1.3_9/OAS/user_projects/domains/java_domain/input` See the *Oracle Retail Store Inventory Management Operations Guide* for details on DexNex batch file processing.
Disable Pack Size	Yes, No	No	This option allows the user to edit the pack size in the application.
EMail From Name	Editable text field	simAlert@myCompany.com	When the system sends e-mail alerts, the specified e-mail address is displayed in the from name.

Table 5–2 (Cont.) Topic: Admin Options

System Administration Option	Valid Values	Default Value	Description
Enable Multiple Set of Books	Enabled, Disabled, SIM Only	Disabled	If the option is **Enabled**, SIM is expected to be in sync with RMS for the purpose of MSOB. If **Disabled**, MSOB functionality is disabled. If **SIM Only**, SIM uses the organization unit to limit the suppliers available to the store.
Enable RSL Integration	Yes, No	Yes	If set to **No** SIM does not attempt to make an RSL connection for both store orders and price change requests.
Online Help URL	Editable text field	na	
RPM/RSL Connection First Name	Editable text	ISOUser	First name of user logging in to RPM.
RPM/RSL Connection Last Name	Editable text	ISOUser	Last name of user logging in to RPM.
RPM/RSL Connection User Name	Editable text field	ISOUser	Description must be a valid RPM username for RSL to work. Used for logging on RPM side.
RSL Timeout (seconds)	Numeric	120	This option specifies the number of seconds before a timeout message is returned.

Table 5–3 Topic: Audit Options

System Administration Option	Valid Values	Default Value	Description
Audit Direct Store Delivery	Yes, No	Yes	Direct store deliveries are tracked with process ID 8 (Direct_ Delivery)
Audit Inventory Adjustment Create	Yes, No	Yes	The creation of inventory adjustments is tracked with process ID 12 (Inventory_Adjustment_Create).
Audit Inventory Adjustment Dispatch	Yes, No	Yes	The dispatch of an inventory adjustment is tracked with process ID 14 (Inventory_Adjustment_Complete).
Audit Inventory Adjustment Update	Yes, No	Yes	The updating of an inventory adjustment is tracked with process ID 13 (Inventory_Adjustment_Update).
Audit Item Requests	Yes, No	Yes	Completed item requests are tracked with process ID 201 (Store_ Order_request).
Audit Price Adjustment	Yes, No	Yes	Price adjustments in the store are tracked by tracking printing of the tickets. Process ID 15 is used (Price_label_printed).
Audit Publish Message	Yes, No	Yes	The publishing of Retail Integration Bus (RIB) messages is tracked using process ID 16 (Publish_Message).
Audit Receive Message	Yes, No	Yes	The subscription to RIB messages is tracked using process ID 17 (Receive_Message).
Audit Return Stock Update	Yes, No	Yes	A return that updates inventory is tracked using process ID 10 (Return_Stock_Dispatch).

Table 5–3 (Cont.) Topic: Audit Options

System Administration Option	Valid Values	Default Value	Description
Audit Security	Yes, No	Yes	Indicates if security information that is created and changed within SIM should be audited. If set to **Yes**, create/edit roles and failed/successful login attempts are audited.
Audit Session Timeout	Yes, No	Yes	A session time-out is tracked with process ID 4 (Session_Timeout).
Audit Stock Count Completed	Yes, No	Yes	Completed stock counts are tracked using Process ID 18 (Stock_ Count_Completed).
Audit Stock Count Processed	Yes, No	Yes	The completion of a recount is tracked using process ID 101 (Stock_ Recount_Completed).
Audit Transfer Dispatch	Yes, No	Yes	When a transfer is dispatched, it is logged using process ID 5 (Transfer_Dispatch).
Audit Transfer Receiving	Yes, No	Yes	When a transfer is received, it is logged using process ID 6 (Transfer_receive).
Audit Transfer Update	Yes, No	Yes	When a transfer is updated, it is logged using process ID 7 (Transfer_update).
Record an audit record for adjust delivery request	Yes, No	Yes	If this option is set to Yes, the system writes an audit record to the database when an adjust delivery request is made.

Table 5–4 Topic: Direct Delivery Options

System Administration Option	Valid Values	Default Value	Description
Add Item to Direct Delivery on Receive	Yes, No	Yes	This option gives the user the ability to add unexpected items when receiving a direct delivery.
Default Direct Delivery Identification Method	Item, Supplier, Purchase Order	Item ID	When starting to identify a direct delivery on the PC, the user can choose to start with the item, supplier, or purchase order number.
Direct Delivery Preferred Currency	Store Currency, Supplier Currency	Store Currency	This option defaults the store or supplier currency to newly created POs depending on preference.
Direct Delivery Send Null Unit Cost	Yes, No	No	If set to Yes, SIM sends a null unit cost to the merchandising system, if the Delivery unit cost equals the Supplier Country unit cost. If set to No, SIM always sends unit cost to the merchandising system.
Disable Supplier Indicator for Purchase Order Creation	Yes, No	Yes	This option allows the system to ignore the **create new purchase order for supplier** flag. If the option is set to **Yes**, the system does not check the flag and always allows stores to create purchase orders. If the option is set to **No**, the system verifies when creating a direct delivery that it is allowed by the supplier.

Table 5–4 (Cont.) Topic: Direct Delivery Options

System Administration Option	Valid Values	Default Value	Description
Display Unit Cost for Direct Deliveries	Yes, No	Yes	This option allows the user to view and edit the unit cost on a direct delivery. Regardless of the display option, the system populates the unit cost for unexpected items or newly created purchase orders in SIM, if a unit cost is available.
Enable DSD Pack Receiving	Yes, No	Yes	This option allows packs to be received in direct store deliveries.
Number of days received direct deliveries can be adjusted	Numeric	0	This option specifies the number of days received direct deliveries can be reopened and adjusted. If a direct delivery falls within the number of days, an **Adjust Delivery** button is displayed on the received delivery. The user can edit values and confirm the delivery.

Table 5–5 Topic: External Finisher Options

System Administration Option	Valid Values	Default Value	Description
External Finisher Enable/Disable	Enable, Disable	Disabled	When this parameter is enabled, the following dialog box displays the finisher details on the PC: ■ Return Filter ■ Return Details ■ Finisher Lookup ■ Warehouse Delivery Filter ■ Finisher Details If enabled, the handheld workflow will display finisher details.

Table 5–6 Topic: Pricing Options

System Administration Option	Valid Values	Default Value	Description
Enable Price Change	Yes, No	Yes	This option allows SIM to make price changes.

Table 5–7 Topic: Purge Options

System Administration Option	Valid Values	Default Value	Description
Days to Hold Audit Records	Numeric	30	Records are deleted in which the create date is less than or equal to the current date minus the number of days to hold.
Days to Hold Completed Customer Orders	Numeric	30	Indicates the number of days that Canceled and Fulfilled Customer Orders will be held in the system before being purged.
Days to Hold Completed Inventory Adjustments	Numeric	30	Records in **Complete** status are deleted, where the inventory complete date is less than or equal to the current date minus the number of days to hold.
Days to Hold Completed Purchase Orders	Numeric	30	All records in **Closed** status are purged after this number of days, where the complete date (the date of when all items were received on the order) is less than or equal to the current date minus the number of days to hold.
Days to Hold Completed Staging Records	Numeric	3	All successfully processed or deleted records will be purged where the update date is less than or equal to the current date minus the days to hold.
Days to Hold Completed Stock Counts	Numeric	30	Purge any records this number of days after the last stock count event has occurred. In other words, when the schedule date is less than or equal to the current date, SIM subtracts the number of days to hold completed stock counts from the date and deletes when this date is reached. A record is purged if the stock count has a status of **Complete**, except in the case of Unit and Amount stock counts. Unit and Amount stock counts are deleted only when the status is Authorize Completed.
Days to Hold Completed UINs	Numeric	120	Indicates how long completed UINs are kept in the system. Completed UINs are defined as any UIN that is in one of the following statuses: ■ Sold ■ Shipped to Warehouse ■ Shipped to Vendor ■ Shipped to Finisher ■ Removed from Inventory ■ Customer Fulfilled
Days to Hold Deleted Users	Numeric	30	This will determine the number of days users with a Deleted status will be held in the system. When the status of the user is updated to Deleted, the system should capture the date in GMT and use that as a basis for determining when the user should be purged from the system. All role, store, password history, and so forth should be purged as well.
Days to Hold Expired User Roles	Numeric	30	Store managers have the ability to assign temporary roles to users in the store. For roles that require an expiration date, after the number of days as defined by this new parameter, the temporary roles should be purged from the user.
Days to Hold In Progress Ad Hoc Stock Counts	Numeric	1	Any ad hoc count with a creation date/time stamp older than this number of days is deleted.
Days to Hold Item Requests	Numeric	30	Records are deleted in which the process date is less than or equal to the current date minus the number of days to hold, for item requests in **Completed** or **Cancelled** status. Any requests in **Pending** status are not purged.

Table 5–7 (Cont.) Topic: Purge Options

System Administration Option	Valid Values	Default Value	Description
Days to Hold Item Tickets	Numeric	15	After this number of days, all records in **Printed** or **Cancelled** status are purged from the database, where the status date is less than or equal to the current date minus the number of days to hold.
Days to Hold Locking Records	Numeric	3	Locking records are sometimes left behind because of system crashes or incorrect logout functionality. After this number of days, these records are deleted.
Days to Hold Pick Lists	Numeric	15	All records in **Complete** or **Cancelled** status are deleted, where the post date is less than or equal to the current date minus the number of days to hold.
Days to Hold Price Changes	Numeric	30	All price change records in **Completed**, **Approved**, and **Rejected** status are purged after this number of days, where the price change effective date is less than or equal to the current date minus the number of days to hold.
Days to Hold Price History	Numeric	90	The LE_HST_ITM_SLS_PRC table is purged so that it contains at least four historical prices for the item/store. This is consistent with the four historical prices the user is able to view in Price History within Item Lookup on the handheld. The Price History table could potentially contain more than four records, because a minimum of one regular price record is required to be held in the database. The purge program needs to restrict these records from being deleted. The Days to Hold Price History parameter allows the user to keep records beyond the four most recent historical prices for this number of days, if desired. Prices in the future are not deleted and are not included in the four historical prices that remain in the database.
Days to Hold Received Shipment Records	Numeric	30	All records in **Complete** and **Canceled** status are purged after this number of days, where the inventory completed date is less than the current date minus the number of days to hold.
Days to Hold Received Transfer Records	Numeric	3	All records in **Received**, **Auto Received**, or **Canceled** status are purged after this number of days, where completed date is less than or equal to the current business date minus the number of days to hold. Transfer requests must also be included in this purge process when the transfer request record is in Cancelled Request, **Completed Approved**, or **Completed Rejected** status
Days to Hold Resolved UIN Exceptions	0 - 999	120	Indicates how long resolved UIN exceptions are kept in the system. The date the exception was resolved is the date the system uses to determine if the exception is ready to be purged.
Days to Hold Returns	Numeric	30	All records in **Dispatched** or **Cancelled** status for Return to Warehouse and Return to Vendor/Supplier are purged after this number of days, where the inventory completed date is less than or equal to the current date minus the number of days to hold.
Days to Hold Sales Posting	0 - 999	24	Indicates how long Point-of-Service transactions uploaded from either Oracle Retail Point-of-Service (ORPOS) or Oracle Retail Sales Audit (ReSA) are kept in the system.

Table 5–7 (Cont.) Topic: Purge Options

System Administration Option	Valid Values	Default Value	Description
Days to Hold Temporary UINs	0 - 999	120	Indicates how long a temporary UIN will be kept in the system. A temporary UIN is any UIN with a status of **None**.
Days to Hold UIN Audit Information	0 - 999	120	Indicates how long UIN audit information is kept in the system. Audit information can be purged for a UIN within the system. The date the audit transaction was captured is used to determine if the record needs to be purged.
Purge Received Transfers	Yes, No	Yes	This option allows received transfers to be purged. This option works in conjunction with the Days to Hold Received Transfers system option.

Table 5–8 Topic: Receiving Options

System Administration Option	Valid Values	Default Value	Description
Disable Damages	Yes, No	No	This option allows the user to receive damages on transfers, direct deliveries, and warehouse deliveries.
Disable Discrepancy Checks in all Receiving	Yes, No	No	When this option is set to **Yes**, the user does not have to go through the discrepancies at the end of receiving on the handheld. This configuration applies to transfer receiving, direct deliveries, and to the item level only on warehouse receiving.

Table 5–9 Topic: Returns Options

System Administration Option	Valid Values	Default Value	Description
Add Item to Return Requests	Yes, No	Yes	This option gives the user the ability to add items to a return request (a return generated by an external system).
Days to send Eail alert before not after date for return requests	Numeric	2	Return requests generated in an external system sometimes require the return to be dispatched to the warehouse before a certain date. This option prompts the recipient of the e-mail the specified number of days before the not after date is reached, if the return was not dispatched.
DSD delivery supplier for RTV	If the **DSD delivery supplier for RTV** system option is set to **Yes**, then the system needs to check both the DSD indicator and the return-allowed indicator. If the **DSD delivery supplier for RTV** system option is set to **No**, then only the return-allowed indicator needs to be validated for supplier returns.	Yes	This new indicator will check to see if the DSD-allowed indicator needs to be set in addition to the return allowed values when creating a supplier return in SIM. **Note:** Regardless of the indicator, SIM should always be able to dispatch the RTV if it was created in an external system.

Table 5–10 Topic: Security Options

System Administration Option	Valid Values	Default Value	Description
Authentication Method	External, Internal, Failover, Partial Failover	Internal	External: An external LDAP system has full control of Authentication and store/role assignments. Internal: SIM has full control of Authentication, user creation and store/role assignments. Failover: Indicates that a hybrid approach will be used for authentication. For each time a user is successfully authenticated in an external system, the user information in SIM will be updated, including password. Then if the external security system is unavailable for authentication, SIM will try to authenticate the user internally with the password that was used during the last successful authentication. Partial Failover: All users and roles are kept in SIM only, but the password can be handled externally. The external password can be cached in case of connection failure, but LDAP retains the master password configuration. SIM will check the password externally. If it cannot be found, SIM will look internally. If LDAP rejects the password, then the assumption that the password is externally controlled.
Duration Before Failed Login Attempts are Reset (in hours)	0 -999	24	If a successful login is not achieved and the duration as defined by this new parameter is reached, the previous failure logins will not be counted toward the maximum number of allowed failure login attempts.
Maximum Days for Temporary Users End Date	0 -999	5	For users that are defined as temporary, this will limit the time before Deactivating a temporary user. If the value is set to **0**, then the temporary user account can have any end date.
Maximum Number of Allowed Failure Login Attempts	0 -999	5	This is the number of failed login attempts a user can have before the user account within SIM will be locked. This parameter has no bearing on the external security system. If the external security system also provides this functionality, whichever value is lower will lock the user out first. If the value is set to **0**, then the user account within SIM will be set to a status of Locked.
Valid Cached User Authentication Duration (in hours)	0 -999	5	When external authentication occurs and the user is successfully authenticated by the external security system the password will be cached in SIM. If the external security system is down and the number of days defined for this parameter have not passed, the password cached in the system will still be valid for the user and can be used to authenticate the user if the Authentication Method is set to Failover. If the value is set to **0**, then the cached authentication information will not be considered valid and will not be used during the failover process.
Valid Cached User Authorization Duration (in hours)	0 -999	48	When external authentication occurs and roles and stores are passed back to SIM, these values will be cached in SIM. If the external security system is down and the number of hours defined for this parameter have not passed, the roles/stores cached in the system will still be valid for the user and can be applied when the user logs in when the Authentication Method parameter is set to Failover. If the value is set to **0**, then the cached authorization information will not be considered valid and will not be used during the failover process.

Table 5–11 Topic: Stock Count Options

System Administration Option	Valid Values	Default Value	Description
Stock Count Display Default Timeframe	Yes, No	No	This option determines whether the Stock Count Default Time Frame value is a selectable option on the stock count screens on both the handheld and the PC.
Stock Count Lockout Days	Numeric	If RMS is not installed, 1 If RMS is installed, set to agree with the RMS value for lockout days	This option specifies the lead time required by RMS to create a unit and value stock count schedule.
Stock Count Null Count Quantity = 0	Yes/No	No	This option determines whether or not items with a null count quantity will be interpreted as a zero quantity for Unit, Problem Line and Ad Hoc stock counts.
Stock Count Sales Processing	Daily Sales Processing, Timestamp Processing	Timestamp Processing	This option determines the kind of sales processing used for sales that are uploaded during the stock count process. Timestamp processing requires sales data to be uploaded with a specific time for every sales transaction. Daily Sales Processing requires the sales file to be uploaded with at least a date, but no time is required. Processing is less accurate with the latter option, and it can cause problems if stock counts are performed during the business day. **Note**: If **Timestamp Processing** is selected, SIM prompts the user with a message: **Timestamp Processing should not be used if your Sales Audit and Merchandise system does not support sales transactions with timestamps. Do you want to continue?** If **Yes** is chosen, timestamp processing is used. If **No** is chosen, Daily Sales Processing is used.
Unguided Stock Count Allow Multiple Users	Yes, No	No	If unguided stock counts are used, this option allows more then one user to scan simultaneously against the same stock count.
Unguided Stock Counts -- Automatic Save	Yes, No	No	If the parameter is set to Yes, the physical timestamp is taken as soon as the item is scanned for the first time, and is saved to the database along with the count value once the user scans the next item on the count. If the parameter is set to No, SIM records the physical timestamp in memory once a user begins counting. The timestamp and count value are posted to the database when the user manually saves the count. With this option, it is assumed the user manually saves the count on a frequent basis, particularly when moving from one area in the store to another.
Updating Stock On Hand	Discrepant Items Only/All Items	Discrepant Items Only	This option determines whether the stock on hand will be updated for all items, both discrepant and non-discrepant items, or for discrepant items only when authorizing a Unit, Problem Line or Ad Hoc stock count.

Table 5–12 Topic:Store Orders Options

System Administration Option	Valid Values	Default Value	Description
Restrict Store Purchase Orders to Store-Orderable Items	Yes, No	Yes	This option prevents items that are not store-orderable from being on regular store purchase orders.

Table 5–13 Topic: Time Zone Options

System Administration Option	Valid Values	Default Value	Description
Daily GMT Batch Run	Yes, No	Yes	Indicates if the batch programs can be run multiple times a day.
Enable GMT for Dex/Nex	Yes, No	No	Dictates whether or not the Dex/Nex data being loaded into the system is in GMT.
Enable GMT for Direct Deliveries	Yes, No	No	Indicates whether or not the Direct Delivery messages published by an external system should have dates in GMT or not.
Enable GMT for Foundation Data	Yes, No	No	Indicates whether or not any foundation data messages being loaded into the system are in GMT.
Enable GMT for Inventory Adjustments	Yes, No	No	Indicates which date/time stamp is used in the inventory adjustment message when it is being published.
Enable GMT for Item Requests	Yes, No	No	Indicates whether or not the Item Request message being published should contain date/time stamps in GMT or not.
Enable GMT for Price Changes	Yes, No	No	Indicates whether price change messages being loaded into the system are in GMT or not. This also determines if the pricing date fields need to be converted when pushing data to a price management application using RSL.
Enable GMT for Receiving	Yes, No	No	Indicates whether or not receiving messages need to be published in GMT or not.
Enable GMT for RTVs	Yes, No	No	Indicates whether or not the RTV message being loaded into the system is in GMT. Likewise, if SIM publishes any RTV message this will determine which date/time stamp is used on the message as well.
Enable GMT for Sales Data	Yes, No	No	Dictates whether or not the sales data being loaded into the system are in GMT.
Enable GMT for Stock Counts	Yes, No	No	Indicates which date/time stamp is used in the stock count message when it is being published.
Enable GMT for Store Orders	Yes, No	No	Indicates whether or not the purchase order messages being loaded into the system has dates in GMT or not. Likewise, if SIM publishes any purchase order message this will determine which date/time stamp is used on the message as well.
Enable GMT for Store Transfers	Yes, No	No	Indicates whether or not the Transfer messages being loaded into the system from an external system has dates in GMT or not. Likewise, if SIM publishes any Transfer messages to an external system this will determine which date/time stamp is used on the message as well.

Table 5–13 (Cont.) Topic: Time Zone Options

System Administration Option	Valid Values	Default Value	Description
Enable GMT for Third Party Stock Counts	Yes, No	No	Indicates whether the date/time stamp in the Third party stock count file (DSLDAT) is in GMT or not.
Enable GMT for Vendor ASN	Yes, No	No	Indicates whether or not the Vendor ASN messages being loaded into the system have dates in GMT or not.
Enable GMT for Warehouse Transfers	Yes, No	No	Indicates whether or not the transfer messages being loaded into the system have GMT dates or not. Likewise, if SIM publishes any transfer message to an external system this will determine which date/time stamp is used on the message as well.

Table 5–14 Topic: Transfer Options

System Administration Option	Valid Values	Default Value	Description
Add Item to Transfer on Receive	Yes, No	Yes	This option gives the user the ability to add unexpected items when receiving a transfer.
Days to hold Dispatched Transfer before sending e-mail alert	Numeric	7	After this number of days, an e-mail alert goes to the sending and receiving stores.
Number of days received transfers can be adjusted	Numeric	0	This option specifies the number of days received transfers can be reopened and adjusted. If a transfer falls within the number of days, an **Adjust Delivery** button is displayed on the received transfer. The user can edit values and confirm the transfer.
Receive Entire Transfer	Yes, No	No	If this option is set to **Yes**, the user can only receive the entire transfer exactly as it was sent.
Transfer Damaged EMail Alert	Yes, No	Yes	If this option is set to **Yes**, an e-mail alert is sent when the receiving store receives goods as damaged.
Transfer Dispatch EMail Alert	Yes, No	Yes	If this option is set to **Yes**, an e-mail alert is sent when the sending store dispatches a transfer.

Table 5–14 (Cont.) Topic: Transfer Options

System Administration Option	Valid Values	Default Value	Description
Transfer Force Close Indicator	No Loss, Sending Loss, Receiving Loss	No Loss	This option determines how SIM handles a short receipt between stores. RMS has a similar system option that needs to be set the same as in SIM. ■ **No Loss.** In this case, the quantity that is short-received on the transfer between stores is added back into the SOH of the sending location. ■ **Sending Loss.** SIM empties the remaining quantity in the transit bucket for the quantity that was not received. This difference is adjusted in the stock ledger by RMS for the sending location. ■ **Receiving Loss.** SIM empties the remaining quantity in the transit bucket for the quantity that was not received. This difference is adjusted in the stock ledger by RMS for the receiving location. There is no real difference between SL and RL from a SIM perspective; in both cases, the missing quantities are written off, but from a financial perspective in RMS, there is a large implication.
Transfer Over/Under EMail Alert	Yes, No	Yes	If this option is set to **Yes**, an e-mail alert is sent when the receiving store receives under/over goods.
Transfer Request Approve EMail Alert	Yes, No	Yes	If this option is set to **Yes**, an e-mail alert is sent when a transfer request has been approved.
Transfer Request Reject E-mail Alert	Yes, No	Yes	If this option is set to **Yes**, an e-mail alert is sent when a transfer request has been rejected.

Table 5–15 Topic: User Interface Options

System Administration Option	Valid Values	Default Value	Description
Display HH Length: Diff 1	Numeric	5	This option sets the display of the first differentiator on the handheld to the specified number of characters. Because of space restrictions, no more than 21 total characters can be displayed on a single line on the handheld. The priority of display is Diff 1, Diff 2, Diff 3, and Diff 4. If Diff 1 takes up 20 characters, only one character of Diff 2 can be displayed.
Display HH Length: Diff 2	Numeric	5	This option sets the display of the second differentiator on the handheld to the specified number of characters. Because of space restrictions, no more than 21 total characters can be displayed on a single line on the handheld. The priority of display is Diff 1, Diff 2, Diff 3, and Diff 4. If Diff 1 takes up 20 characters, only one character of Diff 2 can be displayed.
Display HH Length: Diff 3	Numeric	5	This option sets the display of the third differentiator on the handheld to the specified number of characters. Because of space restrictions, no more than 21 total characters can be displayed on a single line on the handheld. The priority of display is Diff 1, Diff 2, Diff 3, and Diff 4. If Diff 1 takes up 20 characters, only one character of Diff 2 can be displayed.
Display HH Length: Diff 4	Numeric	5	This option sets the display of the fourth differentiator on the handheld to the specified number of characters. Because of space restrictions, no more than 21 total characters can be displayed on a single line on the handheld. The priority of display is Diff 1, Diff 2, Diff 3, and Diff 4. If Diff 1 takes up 20 characters, only one character of Diff 2 can be displayed.
Display Item Description	Long Description, Short Description	Short Description	This option specifies whether the long or short item description is displayed in SIM. This option applies to both the handheld and the PC.
Display Item Image Button	Yes, No	No	If set to **Yes**, an **Image** button displays on the Item Lookup Detail screen. If set to **No**, the button does not display. This allows the retailer to hide the button if the retailer does not support displaying images.
Item Request UI Limit	Not to exceed the value of the UI performance limitation configuration setting.	1500	This option will determine the maximum number of items allowed on an Item Request.
Pick List UI Limit	Not to exceed the value of the UI performance limitation configuration setting.	1500	This option will determine the maximum number of line items allowed on a Pick List. This check will occur in addition to and after the application limits the transaction based on the pick list system parameters.
Problem Line UI Limit	Not to exceed the value of the UI performance limitation configuration setting.	1500	This option will determine the maximum number of line items allowed on a Problem Line stock count.
Search Limit Default for Container Lookup	1 - 999	500	This parameter indicates the default search limit for Container Lookup.

Table 5–15 (Cont.) Topic: User Interface Options

System Administration Option	Valid Values	Default Value	Description
Search Limit Default for Inventory Adjustments	1 - 999	500	This parameter indicates the default search limit for the Inventory Adjustment List screen and Filter.
Search Limit Default for Item Lookup	1 - 999	500	This parameter indicates the default search limit for Item Lookup.
Search Limit Default for Price Changes	1 - 999	500	This parameter indicates the default search limit for the Price Change Filter.
Search Limit Default for Sequencing	1 - 999	500	This parameter indicates the default search limit for the No Location List screen filter.
Search Limit Default for Staged Message Lookup	1 - 999	50	This parameter indicates the default search limit on the Filter screen for Staged Messages.
Search Limit Default for Supplier Lookup	1 - 999	500	This parameter indicates the default search limit for Supplier Lookup.
Search Limit Default for UIN Resolution	1 - 999	500	This parameter indicates the default search limit on the UIN Resolution List Screen and Filter.
Unit and Amount Count UI Limit	Not to exceed the value of the UI performance limitation configuration setting.	5000	This option will determine the maximum number of line items allowed on a Unit and Amount stock count. This check will occur in addition to the Product Group variances values.
Unit Count UI Limit	Not to exceed the value of the UI performance limitation configuration setting.	1500	This option will determine the maximum number of line items allowed on a Unit stock count.

Table 5–16 Topic: UIN Options

System Administration Option	Valid Values	Default Value	Description
Allow Unexpected UINs	Yes, No	Yes	If set to **No**, SIM rejects the UIN in the following situations by prompting the user with an error message (EM14r): ■ Unexpected UIN for a Transfer ■ Inventory adjustment when UIN was not assigned to the current store ■ Stock Counts, when the item was not originally assigned to the current store If set to **Yes**, SIM allows the unexpected UIN for the previous situations and sends an e-mail alert and adds the item to problem line stock count. Completing the transaction updates the UIN Status and updates the store the UIN is assigned to.

Table 5–17 Topic: Warehouse Delivery Options

System Administration Option	Valid Values	Default Value	Description
Add Item to Container on Receive	Yes, No	Yes	This option gives the user the ability to add unexpected items when receiving a warehouse delivery.
Number of days received warehouse deliveries can be adjusted	Numeric	0	This option specifies the number of days received warehouse deliveries can be reopened and adjusted. If a delivery falls within the number of days, an **Adjust Delivery** button is displayed on the received warehouse delivery. The user can edit values and confirm the delivery.
Warehouse Quick Receiving – Automatically confirm ASNs	Yes, No	No	**Yes** for this option specifies that the ASN number is confirmed when all containers have been received automatically, if the ASN is not already confirmed. **No** for this option allows a quick QA check, if required, on the handheld; a value of **Yes** makes this impossible, because the ASN is completed.
Warehouse Quick Receiving Enabled	Yes, No	Yes	This option allows the handheld to receive containers directly, without confirming the ASN number. Each container scanned will be fully received. The ASN number may still need to be confirmed as well.
Warehouse Quick Receiving – Prompt for received containers	Yes, No	Yes	**Yes** for this option means that the user is prompted with an error message if the container is already received. This slows down processing. For either **Yes** or **No**, the container scan does not affect the previously received units.
Warehouse Quick Receiving – Receive missing containers	Yes, No	Yes	This option allows Warehouse Quick Receiving to receive containers after the ASN number is confirmed.

6

Reporting

SIM has the ability to produce reports that retailers can customize to reflect the unique requirements of their business.

Operational Reports

Operational reports are generated from within the functional areas of SIM and include information about pick lists, stock count reports, shipping documentation, and so on. SIM uses a reporting tool when generating these reports in order to provide the user with a report formatting/layout mechanism.

The reporting tool allows the end user to specify the exact data fields to be displayed on the report (although this data is limited to the SIM data that is available for the specific operational report). Modifications to the formatting and data displayed on the report are made using the reporting tool.

SIM provides the user with a way to identify a single default report template to use for each of the different operational reports. When the user generates an operational report from within SIM, the application requests the report template that matches the default specified for that report.

Analytical (and Ad Hoc) Reports

Analytical reports leverage data in SIM for historical analysis. Retailers can develop their own and use these reports to make decisions on key business processes within the store (such as previous days deliveries, number of items replenished on a given day, and so on).

The reporting tool provides the retailer with a report formatting/layout mechanism and allows the user to specify the exact data fields to be displayed on the report. All report metrics and parameters are defined using the reporting tool (although this data is limited to the SIM data that is available for the specific report).

Assumptions

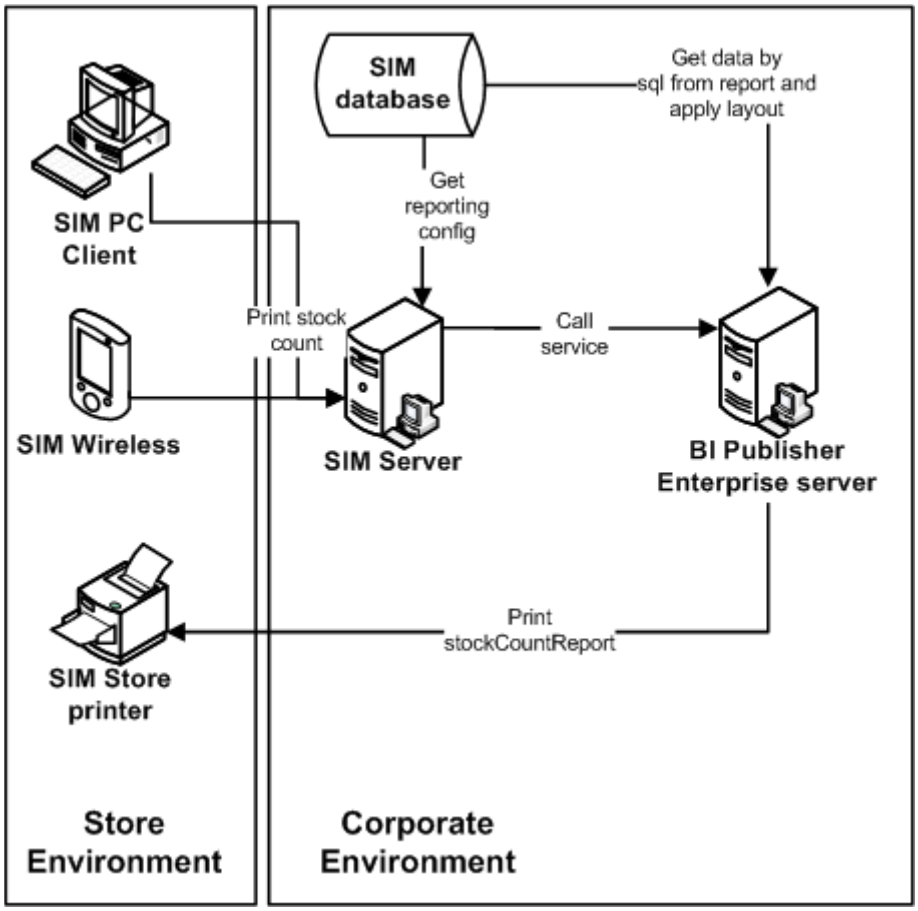
- SIM does not reference any other external security to enforce privileges for the reporting tool. If a user is given the ability to generate a report or to launch the reporting tool within SIM, it is assumed that the user is given the necessary level of access for the reporting tool as well.
- SIM does not manage any scheduling requirements for analytic (and ad hoc) reports. Such scheduling should be handled by the reporting tool itself.

SIM Reporting Framework

Printing to Local Printers in a Store

SIM is able to print to local printers in a store using BIPublisher, as long as the printers are network-enabled printers that support internet protocol printing (IPP). SIM uses BIPublisher as the printing engine. The printers must be network-enabled in order to communicate with BIPublisher.

Figure 6–1 Local Printing in a Store



The following is the workflow for the process of printing locally in a store:

1. A retailer makes a request on the HH or PC to print an item.
2. That request goes to the centrally installed BIPublisher server.
3. The BIPublisher server receives URLs or parameters, runs a query to retrieve data to be printed, and formats the data.
4. That data is then sent back to the store's network-enabled printers.

SIM tables (RK_RETAIL_STORE_PRINTER) refer only to the logical name of the printer. That logical name is setup using BIP-admin, for example, **myPrinter1**. Use BIPublisher to configure **myPrinter1** with specifics.

Do the following to set up a logical printer in SIM with BIPublisher 10.1.3.3 and higher:

1. Set up the logical printer using BIPublisher.
2. Ensure SIM table (RK_RETAIL_STORE_PRINTER) refers to this logical name.
3. Select the **PDF to post-script** option for printing.

SIM Operational Reports

Table 6–1 Operational Reports

Report Name	Report Parameters	Report view/s
ItemDetailReport	ITEMID, STOREID, Store_ Timezone	ITEM_DETAIL_REPORT_V, ITEMDET_ SEQUENCE_V, ITEMDET_ALLOCATION_V
DirectDeliveryReport	Receipt_ID, StoreTimezone	DIRECT_DELIVERY_REPORT_V
ItemRequestReport	Item_Request_Id, Store_ Timezone	ITEM_REQUEST_REPORT_V
PickListReport	Pick_List_ID, Store_Timezone	PICK_LIST_REPORT_V
ReturnReport	Return_ID, Store_Timezone	RETURN_REPORT_V
InventoryAdjustmentReport	Inv_Adj_ID, Store_Timezone	INVENTORY_ADJ_V, RK_INV_ADJ_REASON
StockCountReport	STOCK_COUNT_ID, LOCATION_ID, PHASE, Store_Timezone	STOCK_COUNT_REPORT_V
WarehouseDelivery	ReportDocument_ID, Store_ Timezone	WAREHOUSE_DELIVERY_REPORT_V, WAREHOUSE_DELIVERY_SHIPMENT_V
TransferReport	Transfer_ID, Store_Timezone	TRANSFER_REPORT_V
StockCountRejectedItemReport	STORE_ID	STOCK_COUNT_NOF_V
StoreOrderReport	STORE_ORDER_ID	rk_st_order_print, rk_st_order_li_print, rss_ mini_location
ItemTicketReport1	ITEM_ID, DESCRIPTION, PRICE	No view, report is rendered using pass through parameters
ShelfLabelReport1	ITEM_ID, DESCRIPTION, PRICE	No view, report is rendered using pass through parameters
BolReturnReport	Return_ID	bill_of_lading_return_rep_v
BolTransferReport	Transfer_ID	bill_of_lading_transfer_rep_v

Configuring a Report Printer in SIM

To configure a report printer in SIM, go to Admin > Setup>Printers. A screen opens, displaying a list of retail store printers. Users can add, delete, and edit printers for the user's store.

Uploading Reports

The directory sim/bip_reports holds all SIM reports in .zip format (one .zip file per report). The .zip files are comprised of an .xdo file and an .rtf report template. These .zip files can be readily imported to the BI publisher (BIP) server. All the reports are pre-configured with datasource name BIP-SIM-DATASOURCE. A datasource with this exact name and appropriate jdbc connection string will have to be set up on the BIP server. In addition, all SIM operational reports need to be uploaded to the specific user's folder that is accessing SIM reports. They may also be placed in the Guest folder to provide shared access.

The .rtf templates may be modified or customized as needed using the BI Publisher Template builder plug-in for Word.

Setting up the BI Publisher Server

1. Create a user and assign the BI Publisher Scheduler role, in addition to other reporting roles.
2. Create a new jdbc connection with datasource name BIP-SIM-DATASOURCE.
3. If you will print directly to a printer, create the printer in BIP. This printer server name will be used in RK_RETAIL_STORE_PRINTER.PRINTER_NETWORK_ADDRESS in SIM. If a CUPS server is used, this will be set as <*cups_server_name*>/<*printer_name*> in RK_RETAIL_STORE_PRINTER.PRINTER_NETWORK_ADDRESS.
4. When RK_RETAIL_STORE_PRINTER.PRINTER_NETWORK_ADDRESS is set to **browser** (case insensitive), for any report in SIM, click **Print** to launch the report in a browser instead of printing to a physical printer. This is only possible when using the SIM PC client. This feature is useful in initial stages of implementation, or if it is preferred to view the report in a browser.
5. If RK_RETAIL_STORE.PRINTER_TYPE is set to **1** (for postscript printing), the BI Publisher server will output the report in PDF format. If the PRINTER_TYPE is set to **2** (for ticket printing), the BI Publisher server will output the report in raw XML format (also known as DATA format in BI Publisher). This is to enable printing to a label printer, like Zebra printer, using a Custom Filter in BI Publisher.

Printing Labels and Tickets on a Label Printer This release of SIM was tested for printing labels and tickets on a Zebra label and ticket printer. This is achieved by using ZebraLink Enterprise Connector (ZEC) for Oracle BI Publisher. ZEC intercepts raw XML data coming from Oracle BI Publisher and applies a ZPL (Zebra Programming Language) template to create a ZPL stream understood by Zebra printers.

For testing, BI Publisher was configured to output reports in raw XML format, which was redirected to ZEC to print to Zebra printer. ZEC works with a wide range of Zebra printers.

Oracle BI Publisher: Single Sign-On (SSO) Enabled

SIM integrates with an SSO-enabled Oracle BI Publisher server for reporting. The configuration property **Reporting Tool Request User Realm** for the Store Admin screen property should be used if SSO policy includes a user realm (known as Company Name in SSO terminology); if not, the property should be left as **none**.

<*OSSO_USER*>, <*OSSO_PASSWORD*>, and <*OSSO_USER_REALM*> should be used if the Oracle BI Publisher instance does not have local Oracle BI Publisher users and uses Oracle Single Sign-on instead for security. See Table 6–2, " Setting Up SIM Reports", following.

This is a different login from the SIM application SSO user login, and depends on the enterprise application SSO security settings.

Setting up SIM

Select the Reporting topic on the SIM Store Admin Config screen. The following options need to be set up:

Table 6–2 Setting Up SIM Reports

Option	Value
Reporting Tool Request User name	*<BIP_REPORTS_USER>* or *<OSSO_USER>*
Reporting Tool Request User password	*<BIP_REPORTS_USER_PASSWORD>* or *<OSSO_PASSWORD>*
Reporting Tool Request User realm	**none** (default value) or *<OSSO_USER_REALM>*
Reporting Tool Request URL	http://*<bip_server_host>*:port/*<bip_web_app>*/servlet/scheduler
Reporting Tool Address	http://*<bip_server_host>*:port/*<bip_web_app>*/servlet/report or optionally any reports landing page you have created in BIP server. This URL will be launched in a browser when **Reports** is clicked on the SIM PC client.

Notes: *<BIP_REPORTS_USER>* is the reports user that has been created in BI Publisher server to access SIM reports.

UDA Print Setup

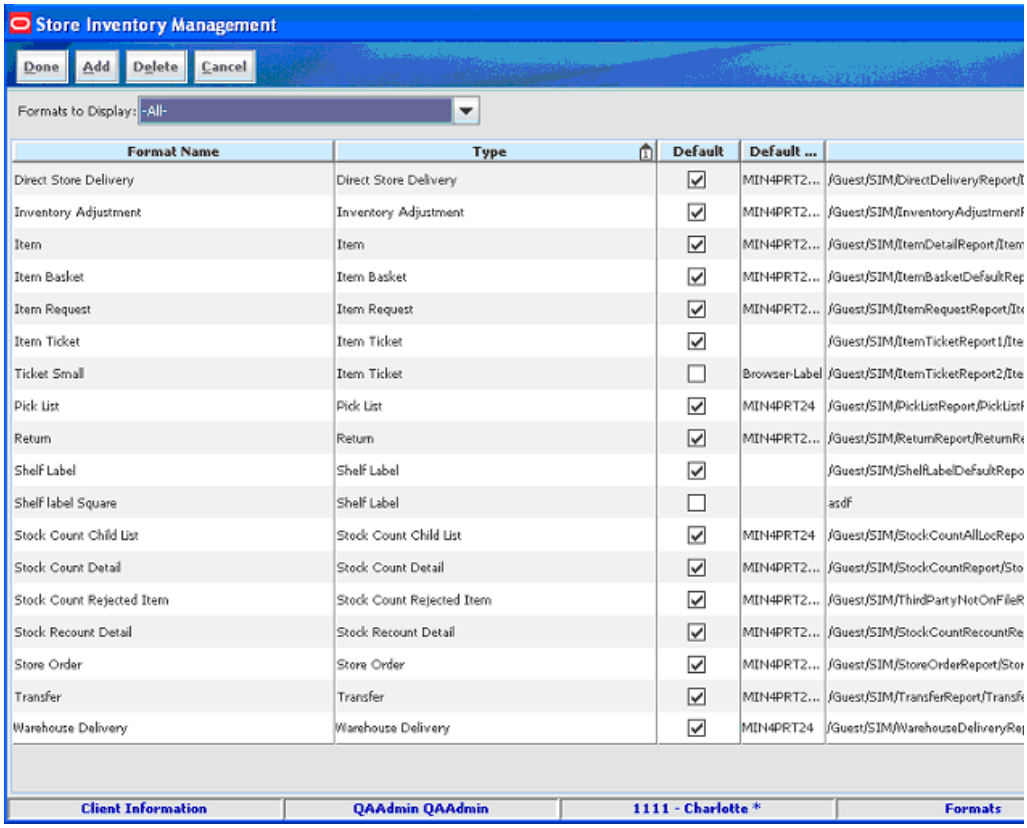
Go to Admin >Setup>UDA Print Setup.

If the UDA is defined on this screen, then when an item UDA is changed, the item UDA is sent to ticketing for a label, ticket or both, based upon definition on this screen.

Setting up Report Formats in SIM

Report Formats are configured in SIM through **Admin> Setup > Formats** screen. See the *Oracle Retail Store Inventory Management User Guide* to add/modify/delete report formats. Multiple formats can be defined for each report type. For a report type, the formats screen enables the user to assign at least one format as the default format.

Figure 6–2 Report Formats Screen



After the SIM reports are uploaded to BI Publisher, the URL location for each report type must be set as follows:

Table 6–3 Report URL Locations

Type	URL Location
Warehouse Delivery	/<*BIP_SIM_REPORTS_FOLDER*>/WarehouseDeliveryReport/WarehouseDeliveryReport.xdo
Transfer	/<*BIP_SIM_REPORTS_FOLDER*>/TransferReport/TransferReport.xdo
Store Order	/<*BIP_SIM_REPORTS_FOLDER*>/StoreOrderReport/StoreOrderReport.xdo
Stock Count Detail	/<*BIP_SIM_REPORTS_FOLDER*>/StockCountReport/StockCountReport.xdo
Stock Count Child List	/<*BIP_SIM_REPORTS_FOLDER*>/StockCountAllLocReport/StockCountAllLocReport.xdo
Stock Count Rejected Item	/<*BIP_SIM_REPORTS_FOLDER*>/StockCountRejectedItemReport/StockCountRejectedItemReport.xdo
Return	/<*BIP_SIM_REPORTS_FOLDER*>/ReturnReport/ReturnReport.xdo
Item Request	/<*BIP_SIM_REPORTS_FOLDER*>/ItemRequestReport/ItemRequestReport.xdo
Item	/<*BIP_SIM_REPORTS_FOLDER*>/ItemDetailReport/ItemDetailReport.xdo
Direct Store Delivery	/<*BIP_SIM_REPORTS_FOLDER*>/DirectDeliveryReport/DirectDeliveryReport.xdo

Table 6–3 (Cont.) Report URL Locations

Type	URL Location
Item Basket	/<BIP_SIM_REPORTS_FOLDER>/ItemBasketReport/ItemBasketReport.xdo
Item Ticket	/<BIP_SIM_REPORTS_FOLDER>/ItemTicketReport1/ItemTicketReport1.xdo
Shelf Label	/<BIP_SIM_REPORTS_FOLDER>/ShelfLabelReport1/ShelfLabelReport1.xdo
Pick List	/<BIP_SIM_REPORTS_FOLDER>/PickListReport/PickListReport.xdo
Inventory Adjustment	/<BIP_SIM_REPORTS_FOLDER>/InventoryAdjustmentReport/InventoryAdjustmentReport.xdo

Note: *<BIP_SIM_REPORTS_FOLDER>* is the folder where SIM reports have been uploaded on the BI Publisher server. For example, if SIM reports have been uploaded to the Guest folder, the folder is **/Guest**.

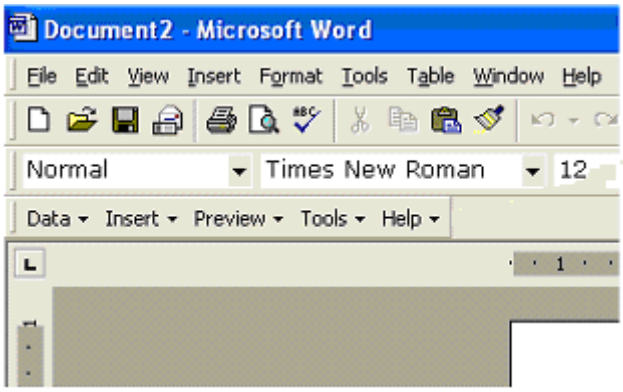
SIM Reports Internationalization

Create a translation (XLIFF) file. For different supported locales, a separate XLIFF file is provided to the BI Publisher. During run time, BI Publisher picks up the default template (RTF) file and the corresponding XLIFF (xlf) file.

Do the following to create an XLIFF file:

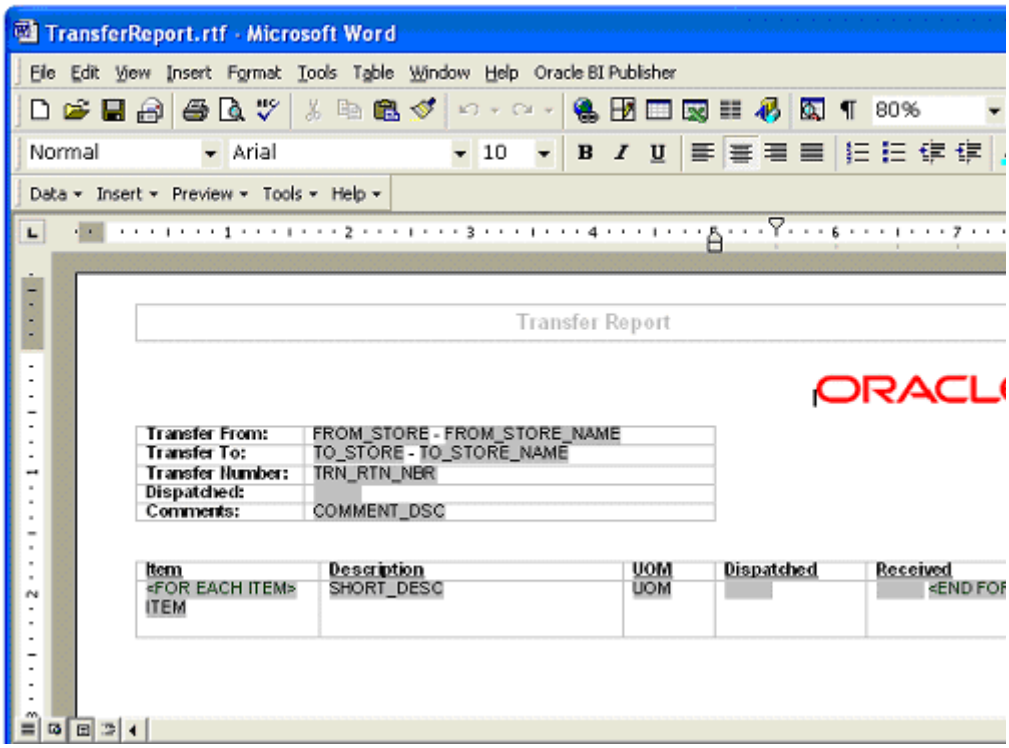
1. Install Oracle BI Publisher Desktop. You see following options in Microsoft Word:
   - Data
   - Insert
   - Preview
   - Tools
   - Help

Figure 6–3 Oracle BI Publisher Desktop Options in Word



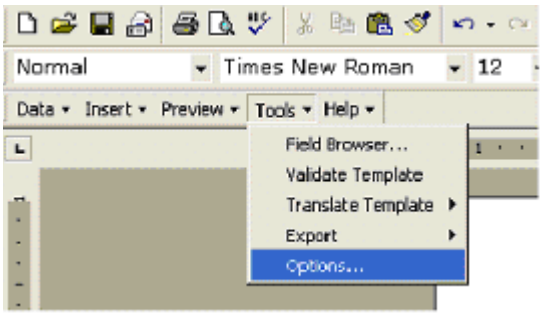
2. Open any existing template in Word, for example, TransferReport.rtf.

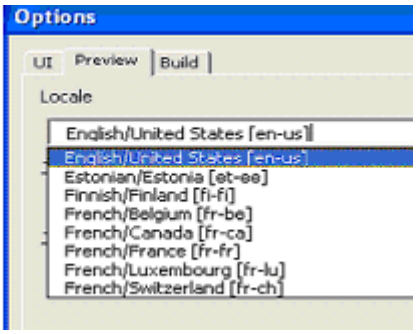
Figure 6–4 TransferReport.rtf



3. Localize the template by selecting **Tools> Option> Preview>Locale**.

Figure 6–5 Localize the Template

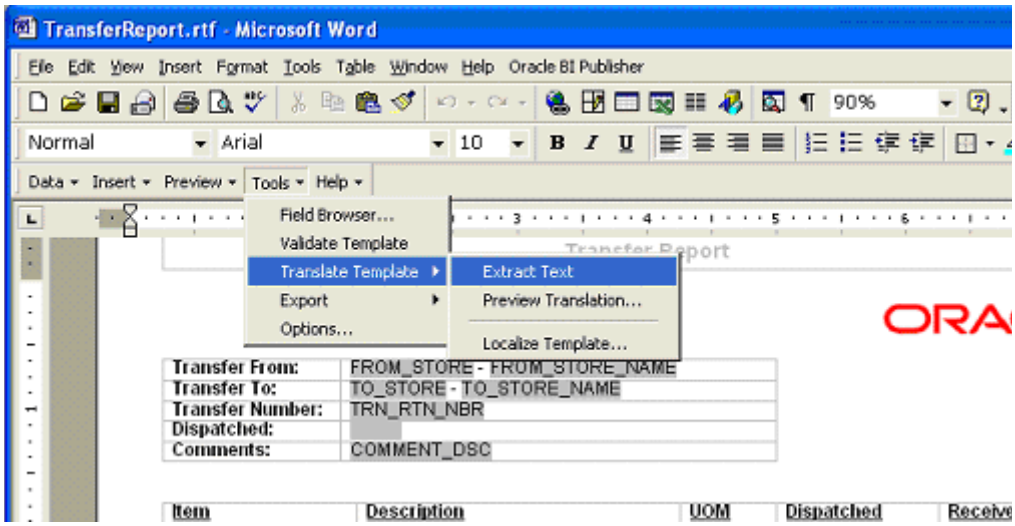




This locale name will appear in the **source-language** attribute of the XLIFF file.

4. From the Template Builder menu, select **Tool>Translate Template>Extract Text**.

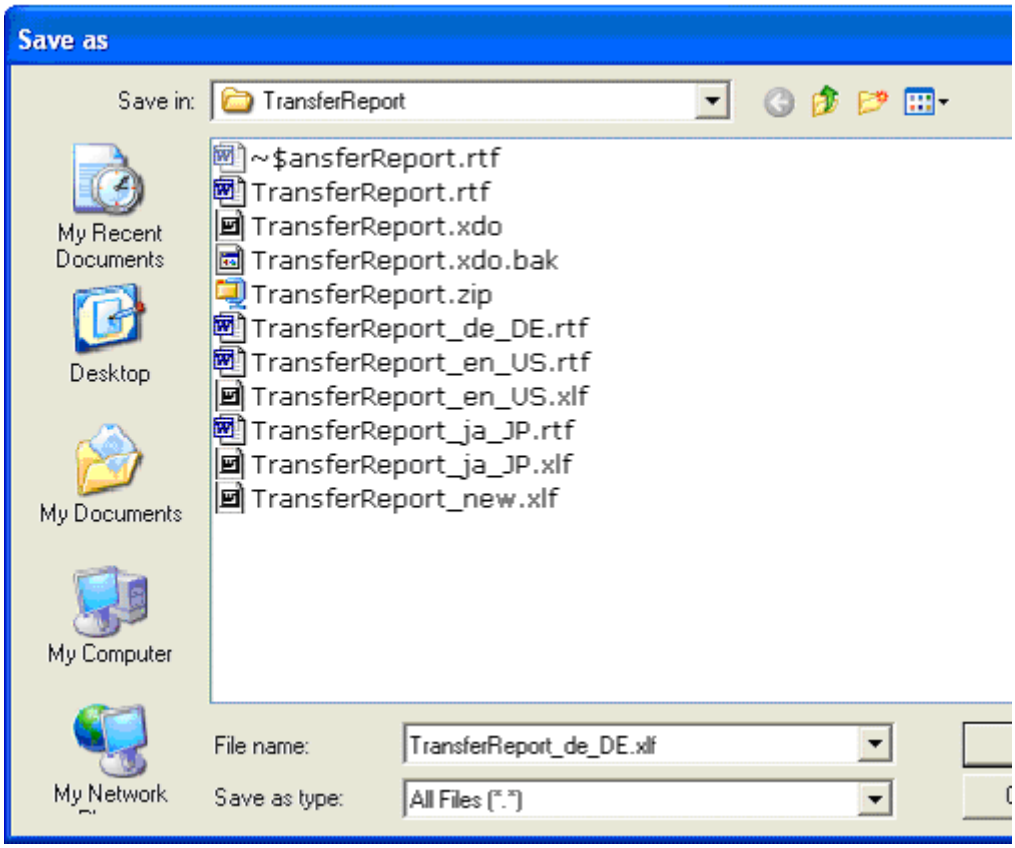
Figure 6–6 Extract Text for Export to XLIFF File



Template Builder extracts the translatable strings from the template and exports them to an XLIFF (.xlf) file.

5. Save the XLIFF file.

Figure 6–7 Save the XLIFF File



6. The XLIFF file generated by XML Publisher has the following structure:

```
<?xml version = '1.0' encoding = 'utf-8'?>
<xliff version='1.0'>
  <file source-language="en_US" target-language="en_US" datatype="XDO"
     original="orphan.xlf" product-version="orphan.xlf" product-name="">
    <header/>
    <body>
     <trans-unit id="e67afb09" maxbytes="4000" maxwidth="23" size-unit="char"
translate="yes">
       <source>Transfer Report</source>
       <target>Transfer Report IN ENGLISH LANGUAGE</target>
       <note>Text located: header/table</note>
     </trans-unit>
     <trans-unit id="7f65664e" maxbytes="4000" maxwidth="23" size-unit="char"
translate="yes">
       <source xml:space="preserve">Printed: </source>
       <target xml:space="preserve">Printed: </target>
       <note>Text located: footer/table</note>
     </trans-unit>
     <trans-unit id="b230538" maxbytes="4000" maxwidth="26" size-unit="char"
translate="yes">
       <source xml:space="preserve">Page Number: [&amp;0] </source>
       <target xml:space="preserve">Page Number: [&amp;0] </target>
       <note>Text located: footer/table</note>
     </trans-unit>
```

The **<file>** element includes the attributes **source-language** and **target-language**.

The valid value for source-language and target-language is a combination of the language code and country code:

- Language Code: the two-letter ISO language code (in lowercase).
- Territory Code: the two-letter ISO country code (in uppercase).

The **<source>** element contains a translatable string from the template in the source language of the template.

The **<target>** element contains the translated string as per locale.

Different XLIFF (xlf) files can be created by providing translated strings to each <target> element and by specifying a target-language value as per naming convention.

Template/XLIFF(xlf) File Locale Selection Logic

At run time, BI Publisher picks up the default template provided in <*ReportName*>.xdo, then applies a translation based on the user's selected Report Locale. First, BI Publisher tries to match an XLIFF file named for the locale, and if an exact match on language-territory is not found, then BI Publisher tries to match on language only.

For example, if you have a report for which the base template is TransferReport.rtf, and the locale is Japanese (ja_JP), then the order of preference in descending order is:

- TransferReport_ja_JP.rtf
- TransferReport_ja_JP.xlf
- TransferReport_ja.rtf
- TransferReport_ja.xlf

- TransferReport.rtf

As soon as BI Publisher finds a matched template (RTF)/XLIFF file, it applies the translation and layout for the report.

Number, Date & Currency Format Support

BI Publisher supports number, date and currency formats by specifying BI Publisher format tasks.

1. Open any existing template in Word, for example, TransferReport.rtf.

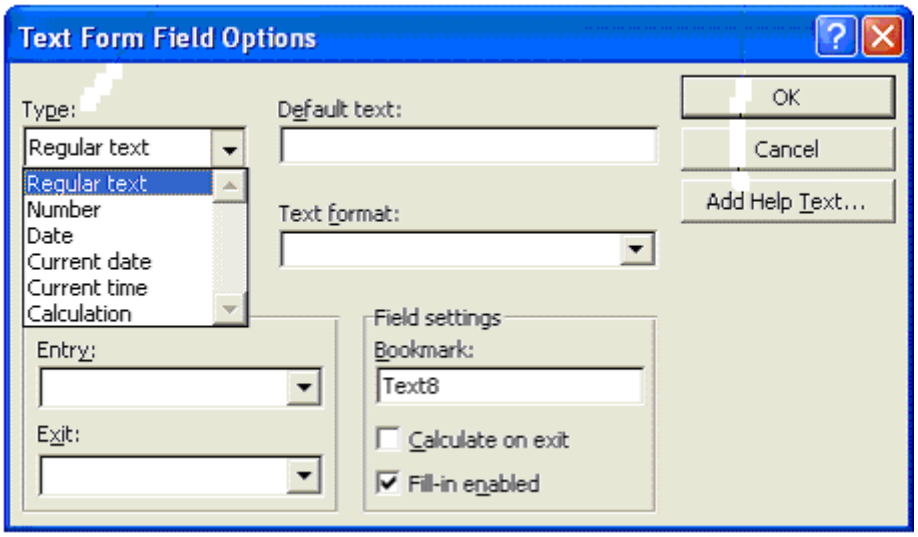
Figure 6–8 Template and Placeholder of the XML Tag

Transfer From:	FROM_STORE - FROM_STORE_NAME
Transfer To:	TO_STORE - TO_STORE_NAME
Transfer Number:	TRN_RTN_NBR
Dispatched:	
Comments:	COMMENT_DSC

Item	Description	UOM	Dispatc
<FOR EACH ITEM> ITEM	SHORT_DESC	UOM	

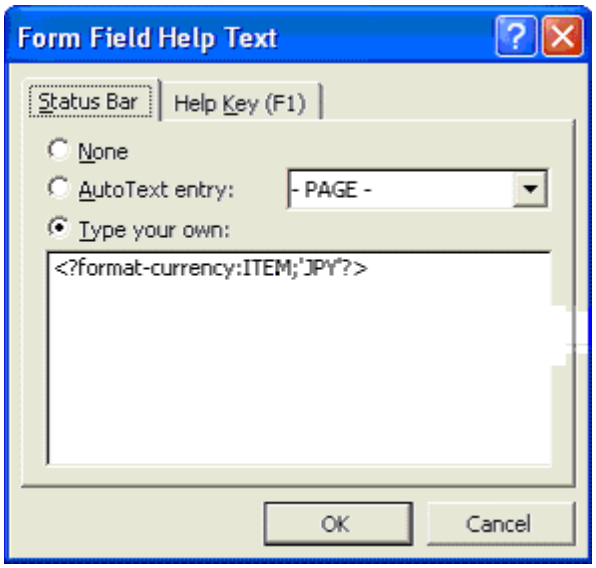
2. Click the **<ITEM>** tag.
3. In the Text Form Field Options window, select **Regular Text** in the Type list.

Figure 6–9 Text Form Field Options Window



4. Click **Add Help Text**.
5. In the Form Field Help Text window, enter the formats for number, currency or date.

Figure 6–10 Form Field Help Text Window



The following are example formats for number, currency and date:

Example 6–1 Number Format

```
<?format-number; 'NUMBER';'999g999D99'?>
```

Where *NUMBER* is the <XML> tag.

Example 6–2 Currency Format

```
<?format-currency;CURRENCY;'CurrencyCode'?>
```

Where *CURRENCY* is the <XML> tag, and *Currency Code* should be ISO specific ('JPY','USD').

Example 6–3 Date Formats

```
<?format-date:date_string; 'ABSTRACT_FORMAT_MASK';'TIMEZONE'?>
```

or

```
<?format-date-and-calendar:date_string; 'ABSTRACT_FORMAT_MASK';'CALENDAR_
NAME';'TIMEZONE'?>
```

Where:

- TimeZone is optional.
- If no format mask is specified, the abstract format mask "MEDIUM" is used as default.

Additional Setting for Currency Format

The following format should be specified in the xdo.cfg file.

Example 6–4 Currency Format in xdo.cfg File

```
<config version="1.0.0" xmlns="http://xmlns.oracle.com/oxp/config/">
    /*********************************************************************/
   <currency-formats>
      <currency code="USD" mask="999D99L"/>
      <currency code="JPY" mask="999D9999X"/>
    </currency-formats>
   /*********************************************************************/
</config>
```

The xdo.cfg file should be uploaded to BI Publisher Server in zipped format along with the xdo, RTF and XLIFF files. See Uploading Reports for more information.

Report Engine Functional Specification

Functional Overview

It is possible on the PC to print multiple reports at the same time to different individual printers or browser sessions.

The handheld will print the report that has been setup as the default option. If no default printer has been set up, the user is prompted to select the printer to print to.

The reporting functionality incorporates error handling when reports are printed. Error handling allows the user to continue in the event that the printing effort fails.

Functional Requirements

SIM Report List

The following reports can be printed:

- Direct Delivery
- Item Request
- Pick List
- Warehouse Delivery
- Returns
- Stock Count/Stock Recount
- Store Order
- Transfers
- Item Report
- Inventory Adjustment
- Item Ticket QR Code Report

Detailed Report Information

Direct Delivery Report

Direct Delivery occurs when the supplier drops off merchandise directly to the retailer's store. This report allows the retailer to print a delivery receipt once all items have been received and the delivery has been finalized.

It consists of the following information broken into three sections:

Header

- Receipt Date–Date on which the receipt was created
- Supplier–Supplier for the PO/ASN received
- Store–Store at which goods were received
- PO Number–PO against which goods were received
- Invoice–Invoice number for the receipt
- Invoice Date–Invoice date for the receipt
- Comments

Detail

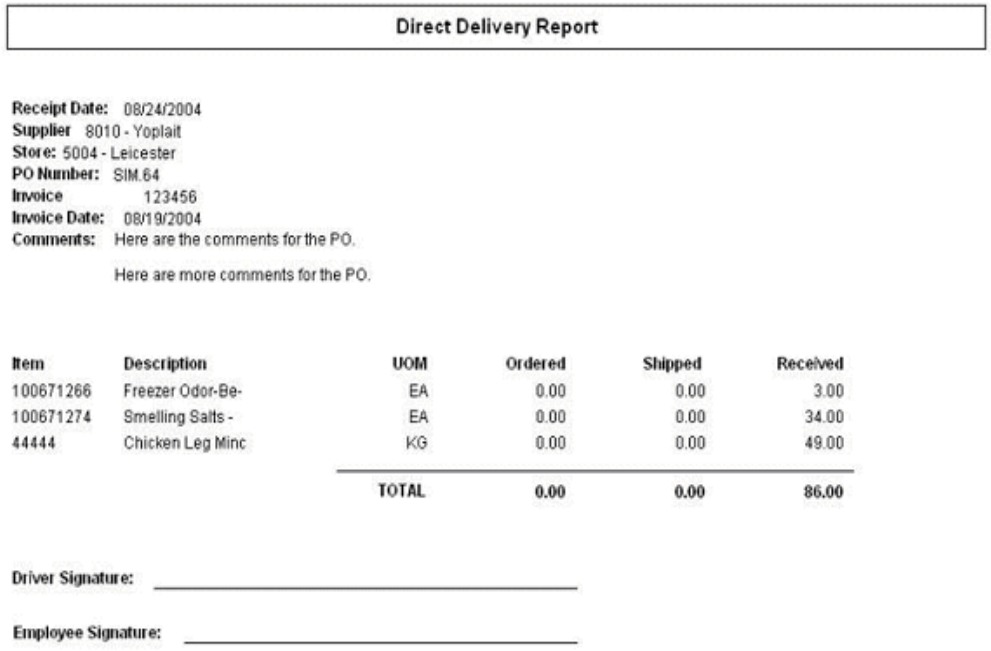
- Item ID–Item number for each line item received
- Item Description–Description of item
- Unit of Measure–Unit of measure for quantity (Cases or Eaches)
- Quantity Ordered–Quantity ordered according to the PO
- Quantity Shipped–Quantity shipped according to the shipment record
- Quantity Received–Quantity actually received
- Unit Cost–Unit cost of the direct delivered item–this column is displayed based on the system parameter (DISPLAY_UNIT_COST_FOR_DIRECT_DELIVERIES) being set

Totals

Totals are provided for the Ordered/Shipped and Received quantities.

A section is also provided as a space holder to collect the signatures of the persons involved in the transaction.

Figure 6–11 Direct Delivery Report



Direct Delivery Report

Receipt Date: 08/24/2004
Supplier 8010 - Yoplait
Store: 5004 - Leicester
PO Number: SIM.64
Invoice 123456
Invoice Date: 08/19/2004
Comments: Here are the comments for the PO.
Here are more comments for the PO.

Item	Description	UOM	Ordered	Shipped	Received
100671266	Freezer Odor-Be-	EA	0.00	0.00	3.00
100671274	Smelling Salts -	EA	0.00	0.00	34.00
44444	Chicken Leg Minc	KG	0.00	0.00	49.00
		TOTAL	**0.00**	**0.00**	**86.00**

Driver Signature: _____

Employee Signature: _____

Item Request Report

The item request functionality allows users to request inventory for individual items to manage stock shortages and increased demand. The requests are processed by the RMS using the replenishment parameters and sourcing information setup in RMS. The report allows the store users to print the details of item requests that have been generated.

The report consists of two sections with the following information:

Header

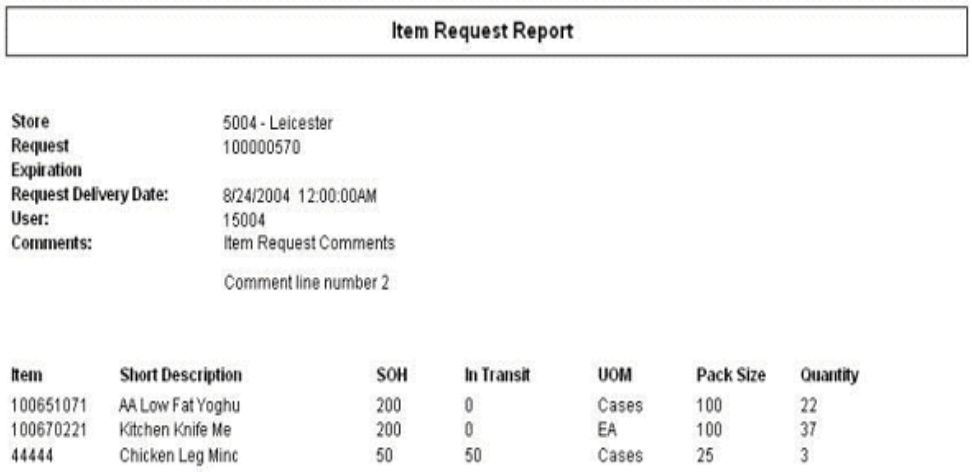
- Store–Store ID and name
- Request ID–Request ID referencing the request in the SIM system
- Expiration Date–Date setup to automatically close item requests that have been automatically generated by the product group scheduler, if no action has been taken
- Request Delivery date–Date on which requested product is wanted at the store
- User–User who generated the item request
- Comments–Additional information

Detail

- Item–Item number for each line item requested
- Short Description–Description of item
- SOH–Current available on hand inventory for the item
- In Transit–Current inventory in transit to the store
- UOM–Unit of measure for the request

- Pack Size–Pack size for the item
- Quantity–Quantity requested

Figure 6–12 Item Request Report



Item Request Report

Store	5004 - Leicester
Request	100000570
Expiration	
Request Delivery Date:	8/24/2004 12:00:00AM
User:	15004
Comments:	Item Request Comments
	Comment line number 2

Item	Short Description	SOH	In Transit	UOM	Pack Size	Quantity
100651071	AA Low Fat Yoghu	200	0	Cases	100	22
100670221	Kitchen Knife Me	200	0	EA	100	37
44444	Chicken Leg Minc	50	50	Cases	25	3

Pick List Report

The pick list report is related to the shelf replenishment functionality supported by SIM. Shelf replenishment in SIM facilitates movement of product between the back room and the shop floor. The pick lists generated list items and quantities that need to be replenished to the shop floor. The pick list report allows the users to print the generated pick list for operational purposes (for example, to use as a reference for the actual picking of product by the store associate).

The report consists of two sections with the following information:

Header

- ID–Pick list identifier used to uniquely identify a pick list
- Product Group–Description for the pick list, based on the product group used to generate the pick list
- Create Date/Time–Date/Time when the pick list was generated
- User–User who generated the pick list
- Status–Current status of the pick list
- Quantity–Total quantity to be picked for the items in the pick list. In case of within day pick lists, the system only adds items until the quantity to be picked is equal to the total quantity entered

Detail

- SKU–Item number for each line item to be picked
- Description–Description of item
- Pick From–Identifies where the product is to be picked from, could be either the back room or the delivery bay
- UOM–Unit of measure for the item to be picked
- Pack Size–Pack size for the item to be picked
- Qty–Quantity of the product to be picked
- Actual Qty–Actual quantity which was picked for the product

Figure 6–13 Pick List Report

SKU	Description	Pick From	UOM	Pack Size	Qty	Actual Qty
100637121	Cheese Knife	Backroom	EA	1	25	25
100670212	Kitchen Knife La	Backroom	EA	1	25	25
100670221	Kitchen Knife Me	Backroom	Cases	100	1	100
100670255	Mixing Bowl	Backroom	Cases	100	1	100
100670263	Tea Towels	Backroom	EA	1	50	50
100670271	Oven Gloves	Backroom	EA	1	50	50
100670301	Salad Bowl	Backroom	Cases	100	1	100
100670319	Measuring Jug	Backroom	EA	1	50	50

Warehouse Delivery Report

Warehouse deliveries in SIM refer to products that are sent from a warehouse to the receiving store. Receiving for warehouse deliveries can be either at the shipment, container, or item level. The warehouse delivery report provides the ability to print details of the warehouse delivery shipments.

The report consists of two sections:

Header

The report header consists of information for the shipment and contains the following information:

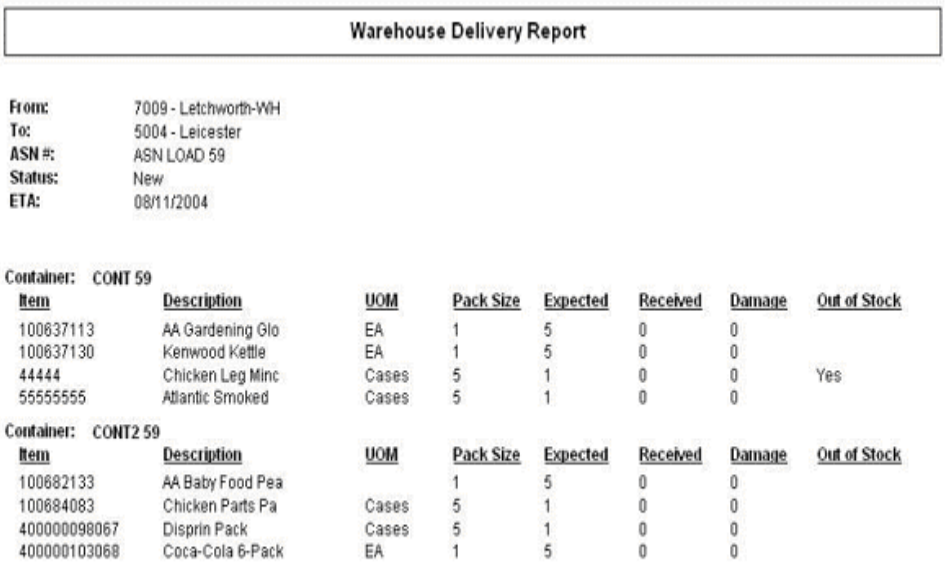
- From–Originating Warehouse location details
- To–Destination store location details
- ASN # - Identifier for the shipment being received
- Status–Status of the warehouse delivery
- ETA–Expected arrival date of the warehouse delivery

Detail

The report detail is broken down by containers included in the shipment and contains the following information:

- Container–References the container label for the items that were shipped. A shipment could consist of multiple containers, in which case the report is grouped by containers
- Item–Item number for each line item requested
- Description–Description of item
- UOM–Unit of measure for the item
- Pack size–Pack size for the item
- Expected–Quantity expected in the container
- Received–Quantity actually received
- Damaged–Quantity marked as damaged
- Out of Stock–Indicates weather the product is currently out-of-stock at the store

Figure 6–14 Warehouse Delivery Report



Warehouse Delivery Report

From: 7009 - Letchworth-WH
To: 5004 - Leicester
ASN #: ASN LOAD 59
Status: New
ETA: 08/11/2004

Container: CONT 59

Item	Description	UOM	Pack Size	Expected	Received	Damage	Out of Stock
100637113	AA Gardening Glo	EA	1	5	0	0	
100637130	Kenwood Kettle	EA	1	5	0	0	
44444	Chicken Leg Minc	Cases	5	1	0	0	Yes
55555555	Atlantic Smoked	Cases	5	1	0	0	

Container: CONT2 59

Item	Description	UOM	Pack Size	Expected	Received	Damage	Out of Stock
100682133	AA Baby Food Pea		1	5	0	0	
100684083	Chicken Parts Pa	Cases	5	1	0	0	
400000098067	Disprin Pack	Cases	5	1	0	0	
400000103068	Coca-Cola 6-Pack	EA	1	5	0	0	

Return Report

The returns functionality allows the store to ship returns either to the warehouse or directly to the vendor. The returns report can be printed as used either as a packing slip for the shipment or as a report for operational records of the store.

The report consists of two sections with the following information:

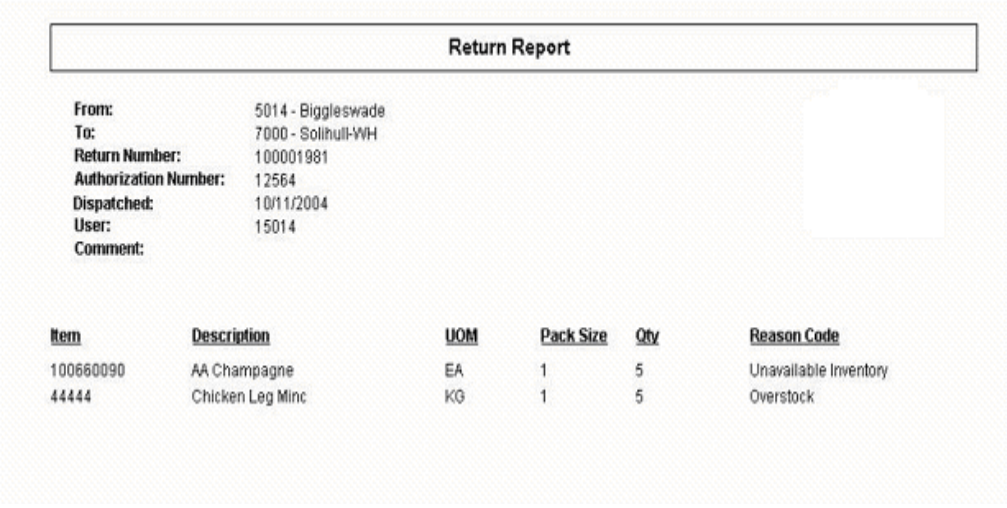
Header

- From–Origin store location and description
- To–Destination location (could be either warehouse or supplier)
- Return Number–Reference number that uniquely identifies the return
- Authorization Number–An authorization number from the vendor referencing the document authorizing the return
- Status/Date–The current status of the return and the date on which the status was changed.
- User–User who created the return
- Not after date–Date after which the return cannot be dispatched (relevant in case of return requests received from the merchandising system)
- Comment–Additional information

Detail

- Item–Item number for each line item on the returned
- Description–Description of item
- UOM–Unit of measure for the item
- Pack Size–Pack size for the item
- Qty–Quantity returned
- Reason Code–Reason code for the return

Figure 6–15 Return Report



Stock Count Report

SIM provides the functionality to schedule, perform and authorize stock counts. The stock counts report provides the store users with the ability to print out scheduled stock counts and use the printed list of record results of the counting on the printed list before entering them into the system.

The report consists of two sections with the following information:

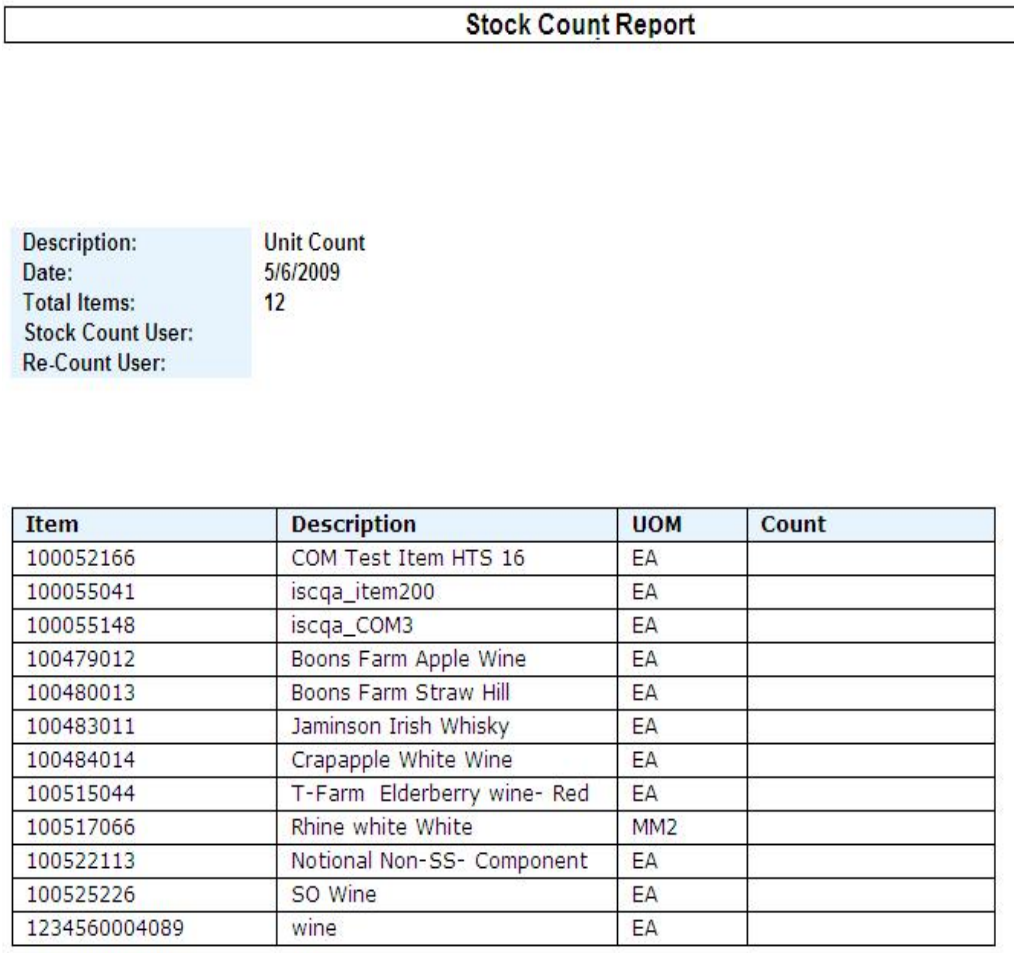
Header

- Description–Master stock count description
- Date–Scheduled date for the master stock count
- Total Items–Total number of items in the master stock count
- Stock count user–User who last saved/completed the stock count
- Recount user–User who last saved/completed the recount

Detail

- Child Count Description–description of the child count appears as a header to the detail section (separate header for each child count). For guided counts, this will be the macro location name along with shopfloor/backroom if sequencing is being used.
- Item–Item number for each line item in the stock count
- Description–Description of item
- UOM–Unit of measure for the item
- Count–Physical count results entered for the stock count

Figure 6–16 Stock Count Report



Stock Count Report

Description:	Unit Count
Date:	5/6/2009
Total Items:	12
Stock Count User:	
Re-Count User:	

Item	Description	UOM	Count
100052166	COM Test Item HTS 16	EA	
100055041	iscqa_item200	EA	
100055148	iscqa_COM3	EA	
100479012	Boons Farm Apple Wine	EA	
100480013	Boons Farm Straw Hill	EA	
100483011	Jaminson Irish Whisky	EA	
100484014	Crapapple White Wine	EA	
100515044	T-Farm Elderberry wine- Red	EA	
100517066	Rhine white White	MM2	
100522113	Notional Non-SS- Component	EA	
100525226	SO Wine	EA	
1234560004089	wine	EA	

Private and Confidential

Stock Count Re-Count Report

SIM allows the store users to create and schedule stock counts that will trigger an automatic recount when the counts fall outside a pre-defined variance. In case a recount is triggered the stock count recount report provides store users the ability to print out the stock counts that need to be recounted and record the results of the recounts.

The report consists of two sections with the following information:

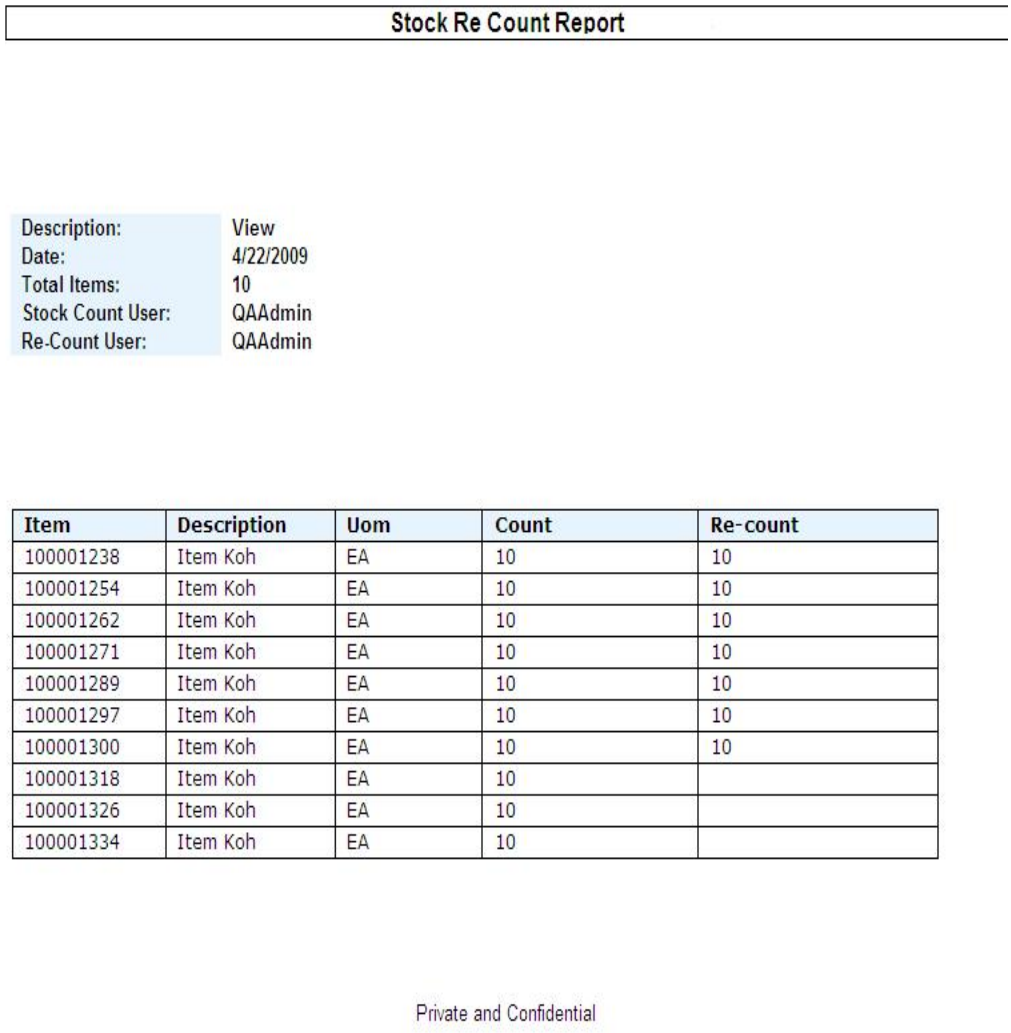
Header

- Description–Master stock count description
- Date–Scheduled date for the master stock count
- Total Items–Total number of items in the master stock count
- Stock count user–User who last saved/completed the stock count
- Recount user–User who last saved/completed the recount

Detail

- Child Count Description–description of the child count appears as a header to the detail section (separate header for each child count). For guided counts, this will be the macro location name along with shopfloor/backroom if sequencing is being used.
- Item–Item number for each line item in the stock count
- Description–Description of item
- UOM–Unit of measure for the item
- Count–Physical count results entered for the initial stock count
- Recount–Count results for the recount of the stock count

Figure 6–17 Stock Count Re-Count Report



Stock Re Count Report

Description:	View
Date:	4/22/2009
Total Items:	10
Stock Count User:	QAAdmin
Re-Count User:	QAAdmin

Item	Description	Uom	Count	Re-count
100001238	Item Koh	EA	10	10
100001254	Item Koh	EA	10	10
100001262	Item Koh	EA	10	10
100001271	Item Koh	EA	10	10
100001289	Item Koh	EA	10	10
100001297	Item Koh	EA	10	10
100001300	Item Koh	EA	10	10
100001318	Item Koh	EA	10	
100001326	Item Koh	EA	10	
100001334	Item Koh	EA	10	

Private and Confidential

Store Order Report

Store orders provide the store users the ability to create and approve orders to a supplier or transfer requests to the warehouse directly in the merchandising system. The store orders report allows the users to print out the report of the order that had been created from the store.

The report consists of two sections with the following information:

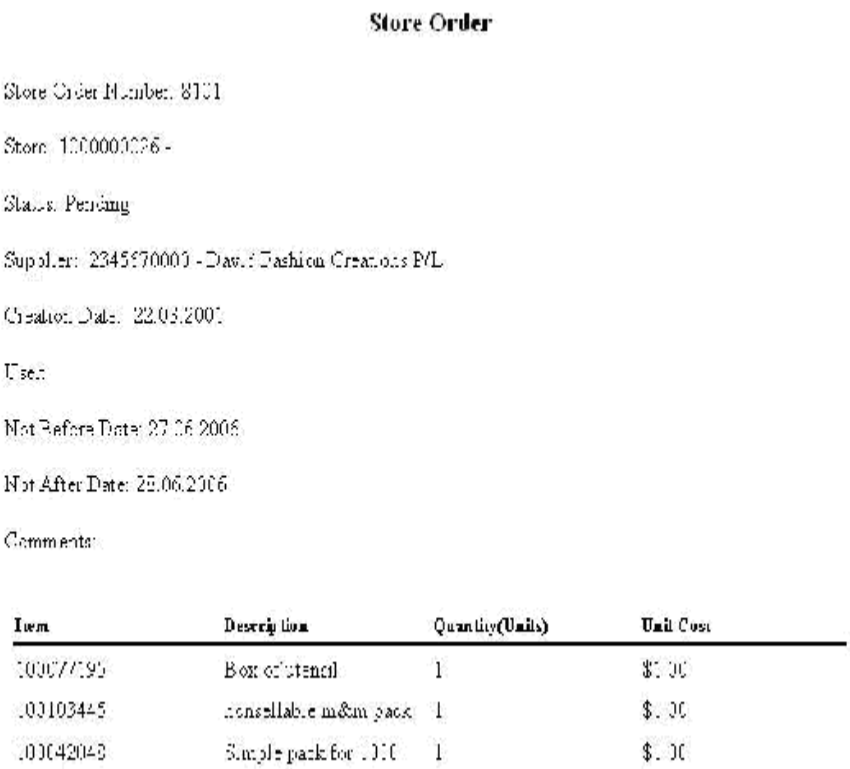
Header

- Store–Store requesting the order
- Store Order Number–Unique reference ID in SIM for the store order
- Status–Current status of the store order. Valid values are **Pending**, **Approved** and **Cancelled**
- Supplier/Warehouse–Source location for the store order
- Creation Date–Date on which the store order was created
- Not before date–Earliest date on which the order can be delivered at the store
- Not after date–Expiration date for the order
- User–User who created the store order
- Comments–Additional information

Details

- Item–Item number for each line item in the store order
- Description–Description of item
- UOM–Unit of measure for the item (part of the quantity heading)
- Qty–Requested quantity for the item
- Unit cost–Unit cost of the requested item

Figure 6–18 Store Order Report



Store Order

Store Order Number: 8101

Store: 1000000036 -

Status: Pending

Supplier: 2345970000 - David Fashion Creations P/L

Creation Date: 22.05.2001

User:

Not Before Date: 27.06.2006

Not After Date: 28.06.2006

Comments:

Item	Description	Quantity(Units)	Unit Cost
10007795	Box of stencil	1	$1.00
10103445	consellable m&m pack	1	$1.00
10042043	Simple pack for 1010	1	$1.00

Transfer Report

Transfer functionality allows stores to transfer stock from one store to another within a company. The transfer report allows the store users to print out the details of either a transfer or a transfer request. The printed report can be used either as a dispatch slip for the transfer shipment or for the store records.

The report consists of two sections with the following information:

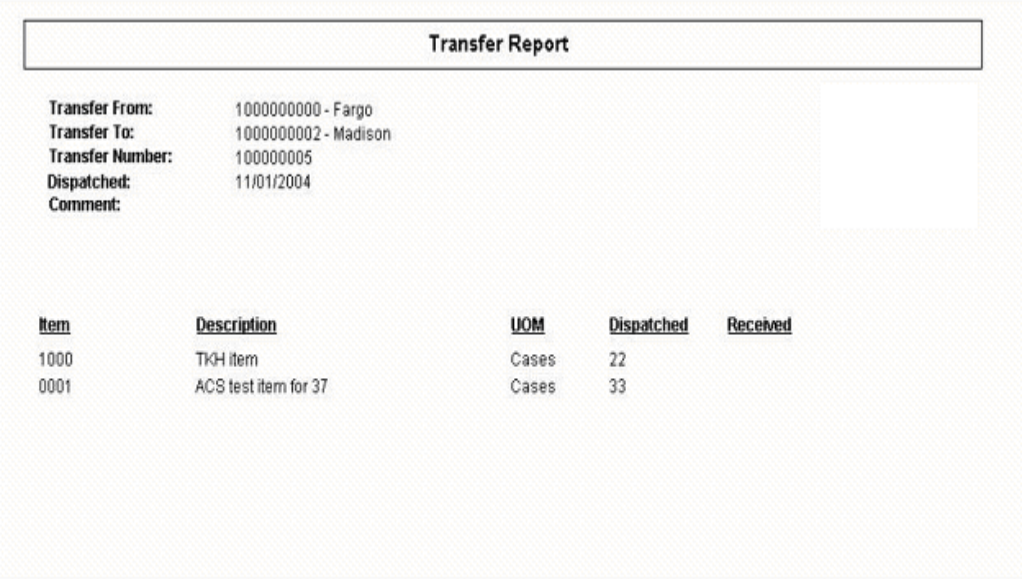
Header

- Transfer from–Origin store location for the transfer
- Transfer to–Destination store location for the transfer
- Transfer number–Unique reference number for the transfer
- Status/Date–Status of the transfer and the date on which the status changed
- Comment–Additional information
- Dispatched–Date on which the transfer was dispatched

Details

- Item–Item number for each line item in the transfer
- Description–Description of item
- UOM–Unit of measure for the item
- Dispatched–Quantity of product dispatched
- Received–Quantity of product received

Figure 6–19 Transfer Report

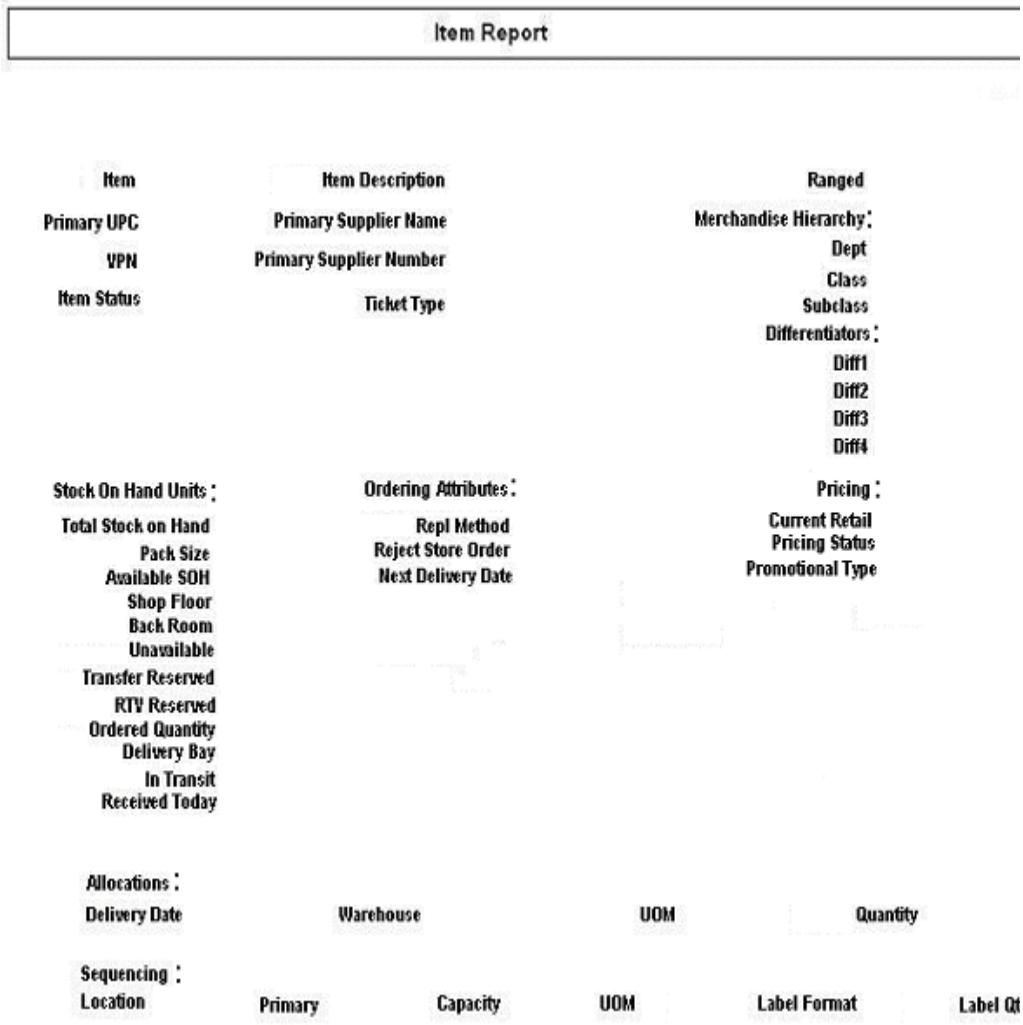


Item Report

This is an Items report that is printed from the item detail screen and the handheld. The report is based on the view Itemlocstock. This report displays the following information:

- SKU number/UPC
- Description (long or short depending on parameter setting)
- Diffs (if any)
- Merchandise Hierarchy
- Inventory Position (Available, unavailable, SOH, reserved)
- Current price
- Forward looking Delivery information (In transit, on order)

Figure 6–20 Item Report



Inventory Adjustment Report

The inventory adjustment report allows the user to select an item that has been adjusted, and print information out for this. The report could be used to help as reference why inventory is unavailable (for example, loaning out for a demo or photoshoot), and confirmation that someone has ownership of that item.

The report consists of two sections with the following information:

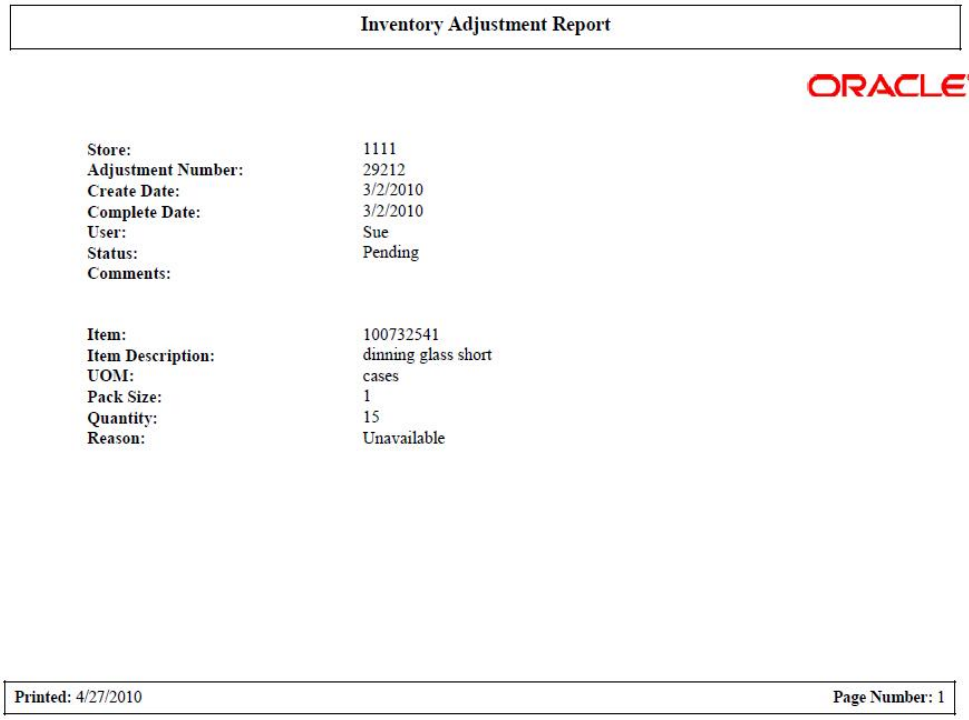
Header

- Store–The store ID
- Adjustment Number–Unique inventory adjustment number in SIM
- Create Date–Date of creation
- Complete Date–Date of completion
- User–User requesting inventory adjustment
- Status–Status **Pending** or **Completed**
- Comments–Comments, if any

Details

- Item–Item number
- Item Description–Item description
- UOM–Unit of measure
- Pack Size–Pack size
- Quantity–Quantity adjusted
- Reason–Inventory adjustment reason

Figure 6–21 Inventory Adjustment Report


Inventory Adjustment Report

ORACLE

Store:	1111
Adjustment Number:	29212
Create Date:	3/2/2010
Complete Date:	3/2/2010
User:	Sue
Status:	Pending
Comments:	

Item:	100732541
Item Description:	dinning glass short
UOM:	cases
Pack Size:	1
Quantity:	15
Reason:	Unavailable

Printed: 4/27/2010 **Page Number:** 1

Item Ticket QR Code Report

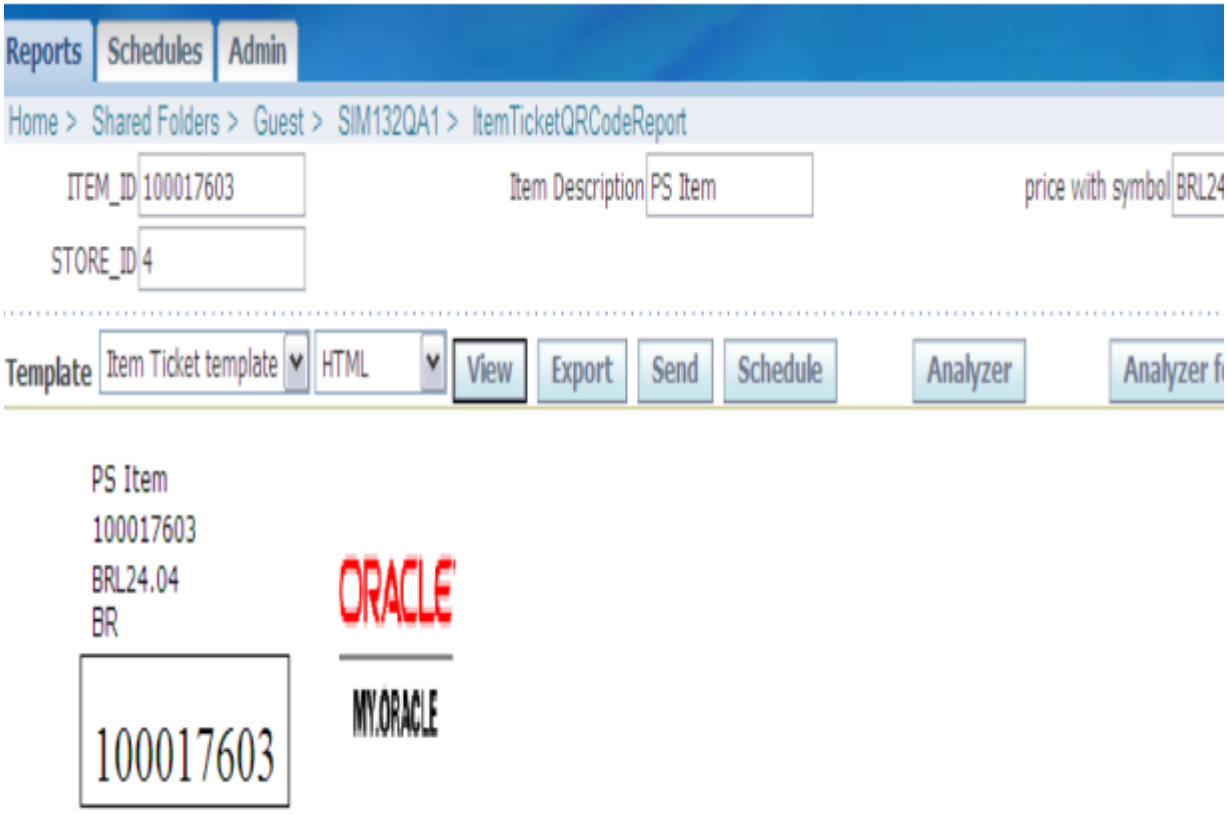
Item Ticket QR Code Report allows the users to create Item Ticket containing the QR code image. BI publisher connects to the Image Server and loads the image at the run time.

The report consists of the following information:

Detail

- Item Description–Description of the item.
- Item_ID–Item number for which Item Ticket is generated.
- Price–The Label price for the item.
- Country of Manufacture–Country of manufacture of the item.

Figure 6–22 Item Ticket QR Code Report



Bill of Lading Report

SIM users can print the Bill of Lading. The printed Bill of Lading displays the status of the Bill of Lading through the appearance or absence of watermarks on the Bill of Lading. A generic template for the Bill of Lading is created and inserted into the format table.

The Bill of Lading report process is designed to allow retailers to create a Bill of Lading report during a Returns or a Transfer dialogue and print it.

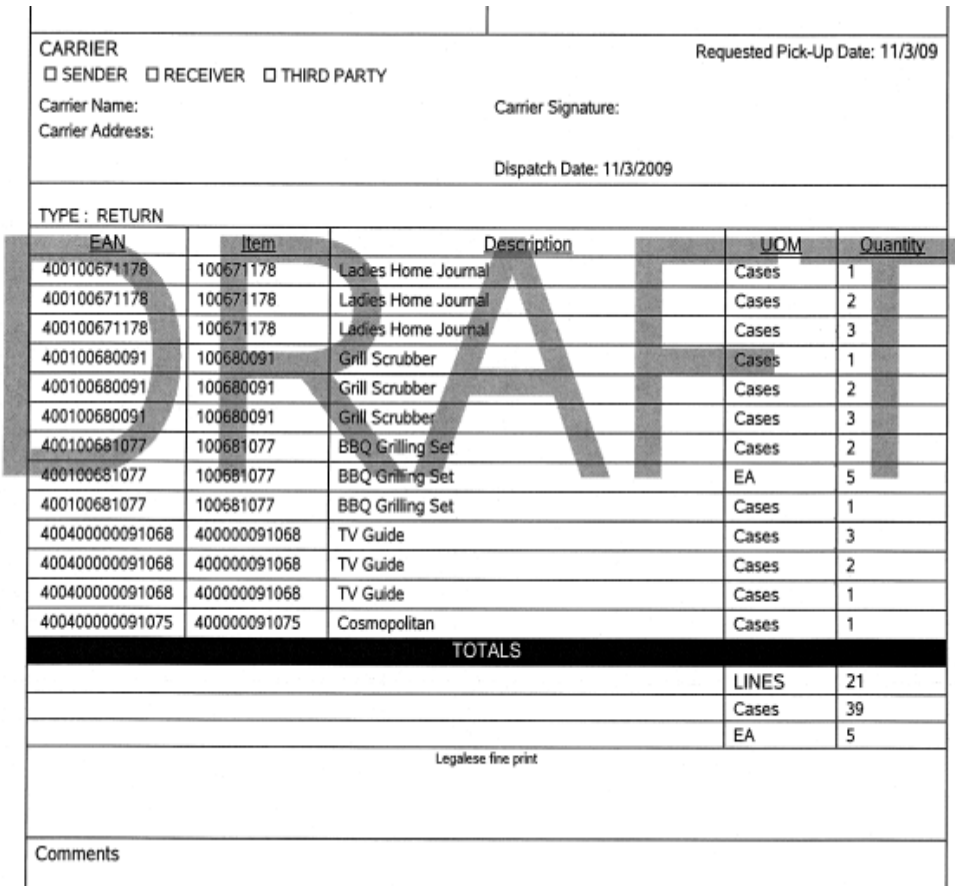
The Bill of Lading consists of information that identifies the sender, the receiver and the carrier of the goods. It also lists the goods and their quantities that are being shipped. It identifies if the shipment is a result of a return or a transfer.

The process consists of two parts, creating the Bill of Lading and printing the Bill of Lading.

The Bill of Lading creation occurs in two stages. During the first stage, the Bill of Lading is automatically created when a shipment is created and saved. During the second stage, the Bill of Lading is updated when the retailer adds additional information to the transaction for the purpose of the Bill of Lading report such as requested pickup date, change of destination address, carrier name and address and change of motive for the shipment. Detailed information for a Return or Transfer such as the quantity being shipped or the items being shipped can be modified at any time prior to the dispatching of a shipment. The Bill of Lading is updated to reflect these changes.

The second part of the process occurs when the retailer prints the Bill of Lading. The Bill of Lading can be printed at anytime during the creation of a transfer or return. It is printed from the Return List screen or the Transfer List screen. It can be printed after the Return or Transfer is canceled (status = **canceled**) or after the transfer is dispatched. Each of these scenarios results in a variation of the Bill of Lading report. For example, if a Bill of Lading is printed prior to dispatching, the Bill of Lading will have a watermark across the page that reads **DRAFT**.

Figure 6–23 Bill of Lading -- Draft Example



CARRIER

☐ SENDER ☐ RECEIVER ☐ THIRD PARTY

Requested Pick-Up Date: 11/3/09

Carrier Name:

Carrier Address:

Carrier Signature:

Dispatch Date: 11/3/2009

TYPE : RETURN

EAN	Item	Description	UOM	Quantity
400100671178	100671178	Ladies Home Journal	Cases	1
400100671178	100671178	Ladies Home Journal	Cases	2
400100671178	100671178	Ladies Home Journal	Cases	3
400100680091	100680091	Grill Scrubber	Cases	1
400100680091	100680091	Grill Scrubber	Cases	2
400100680091	100680091	Grill Scrubber	Cases	3
400100681077	100681077	BBQ Grilling Set	Cases	2
400100681077	100681077	BBQ Grilling Set	EA	5
400100681077	100681077	BBQ Grilling Set	Cases	1
400400000091068	400000091068	TV Guide	Cases	3
400400000091068	400000091068	TV Guide	Cases	2
400400000091068	400000091068	TV Guide	Cases	1
400400000091075	400000091075	Cosmopolitan	Cases	1
TOTALS				
			LINES	21
			Cases	39
			EA	5

Legalese fine print

Comments

DRAFT

If the Bill of Lading is printed after dispatching it will have no watermark on it. If the Bill of Lading is printed after the Transfer or Return has been deleted (status = **canceled**), the Bill of Lading will have a watermark across the page that reads **Canceled**.

Printing the Return Bill of Lading

Print the Bill of Lading from the Return List screen.

The default filter for the Return List screen will be changed to include displaying dispatched returns with the current session.

1. If the return is in the In Progress state, the Bill of Lading prints with the watermark **DRAFT** on each page of the Bill of Lading
2. If the return has been dispatched and it no longer appears on the List screen. use the filter to display the dispatched return.

   The Bill of Lading prints without any watermarks.
3. If the return has been cancelled, use the filter to display the cancelled return.

   The Bill of Lading will print with the watermark **CANCELED** on each page of the Bill of Lading.

Printing the Transfer Bill of Lading

The Bill of Lading can be printed from the Transfer List screen. If the Transfer does not appear on the List Screen, the filter must be used to display the Transfer.

1. If the transfer is in the In Progress state, the Bill of Lading prints with the watermark **DRAFT** on each page of the Bill of Lading.
2. If the transfer is in the Dispatched state, the Bill of Lading prints without any watermarks.
3. If the transfer has been cancelled, the Bill of Lading prints with the watermark **CANCELED** on each page of the Bill of Lading.

7

Customization

Retailers often modify retail software either in-house or have it modified through third-party system integrators. If the customization efforts are not done properly, the deployed product might not operate correctly. A poorly customized product is difficult to upgrade and deploy. This situation causes serious issues for the retailer, Oracle Retail, and any system integrators involved. This chapter aims to mitigate that risk by providing guidance on how to customize safely and effectively.

Build/Packaging/Deployment Related

To build and deploy customizations correctly, make a separate project for custom-written code. Build a custom JAR with the newly created code. This custom JAR should be placed first in the execution classpath so that classes found within the JAR are chosen by the JVM rather than the base classes. This requires un-signing and re-signing all the JARs in the Oracle Retail Store Inventory Management (SIM)-client application, since all JARs must be signed with the same signature for Web Start to work correctly. Consult the jarsigner documentation from Sun for further information on the JAR un-signing/signing process.

Architecture

Do not connect directly to the RMS or SIM database using JDBC or other database connection software from any client layer, whether that is the PC client, Web Service, or Wavelink server layer. This will lead to being unable to guarantee data integrity and thus SIM can no longer be guaranteed to work.

Process Overview

In order to outline reasonably safe approaches to adding new functionality, this document contains three examples of extensions to the base source code:

- Adding new attributes to pre-existing data and workflows. This includes how to update the database, business and client layers.
- Adding completely new data and workflows to the application including modifying all layers.
- Modifying incoming and outgoing data transport to external systems.

Adding Attribute to Pre-Existing Data & Workflows

To add the attribute **Season** to the current Item object and keep the system open to upgrade without difficulty, do the following:

1. Add the New Attribute to the Database
2. Update DAO (Database Access Objects)
3. Update Business Object or Value Object
4. Update the Services
5. Update the PC Screen
6. Update the HH Forms

Add the New Attribute to the Database

Database Tables

There are two standard categories of database tables in SIM: ARTS tables and SIM-specific tables. ARTS tables are based on industry standard definitions. It might be helpful to know that to decode the names of an ARTS table, read the table name backwards. For example, the PA_STR_RTL table represents Retail Store Party. SIM-specific tables either begin with the RK_ prefix to indicate that they are not part of the ARTS model, or they are named in a forward manner such as STOCK_COUNT or STOCK_COUNT_LINE_ITEM.

Create New Attribute Column

Add the appropriate attribute to SIM's current database table or make a new table to store the attribute. Do not alter or remove any columns as this will render SIM unable to access the table.

SIM's item master is AS_ITM, so add a column to the AS_ITM table that is named GRI_SEASON. Naming should include a unique tag from the company customizing the code in order to easily tell the difference between base and customization for upgrade purposes. If it is decided to add a new item attributes table where SEASON was just one of the attributes, then the name of the table might be GRI_ITEM_ATTRIBUTE. If the entire table is custom, it is then appropriate to simply use the column name SEASON.

Update DAO (Database Access Objects)

Create DAO Code to Access Information

SIM will not automatically access this new data in the database. Custom DAO (Data Access Object) code must be written to retrieve this information. It is advisable to always create completely new DAO source code for this customization.

You may choose to write data access layer code to retrieve these properties in whatever manner you choose to. However, if you want to use SIM's frameworks for data retrieval and database connection pooling, declare the new data access layer class and have it extend BaseOracleDao:

```
public class ItemAttributeOracleDao extends BaseOracleDao {
```

BaseOracleDao supplies several helpful methods to perform a lot of the heaving lifting of data access. It enables the user to execute() statements, batch statements, and stored procedures as well as perform several different types of queries. Becoming familiar with the APIs in this class will help in the process of developing database layer code.

Performing Simple Insert

To perform a simple insert, call the execute() method with a ParametricStatement that contains your insert SQL.

```
String insertSQLString = "insert into GRI_ITEM_ATTRIBUTE (ITEM_ID, SEASON) values
(?, ?)";

List<Object> insertSQLParams = CollectionUtil.newArrayList();
insertSqlParams.add(itemId);
insertSqlParams.add(season);

execute(new ParametricStatement(insertSQLString, insertSQLParams));
```

A ParametricStatement is a useful object within the SIM code. It is constructed with the insert SQL string and a list of objects to substitute in the SQL string where "?" appears. Performing a simple update is performed as in the previous example.

Performing Simple Query

To perform a simple query, call the query() method with a ParametricStatement that contains your select SQL:

```
String querySql = "select ITEM_ID, SEASON from GRI_ITEM_ATTRIBUTE where ITEM_ID =
?";

List<Object> querySqlParams = CollectionUtil.newArrayList();
querySqlParams.add(itemId);

List<GriItemAttributeDataBean> beans
      = query(new GriItemAttributeDataBean(),querySql, querySqlParams);
```

Using the query helper method of BaseOracleDao will require creating a data bean. A data bean is a simple class that contains the attributes of the table in question. The data bean implements some required API in order to let the framework execute the SQL and then populate the beans with the correct information.

Creating a Data Bean

To create a data bean, create a class that extends BaseDataBean. Implement both the methods defined by the superclass interface:

```
public class GriItemAttributeDataBean extends
BaseDataBean<GriItemAttributeDataBean> {
    private String itemId;
    private String season;
    public String getSelectSql() {
       return "select ITEM_ID, SEASON from GRI_ITEM_ATTRIBUTE where ITEM_ID = ?"
    }
    public GriItemAttributeDataBean read(ResultSet resultSet) throws SQLException
{
```

```
        GriItemAttributeDataBean bean = new GriItemAttributeDataBean ();
        bean.itemId = getStringFromResultSet(resultSet, "ITEM_ID");
        bean.season = getStringFromResultSet(resultSet, "SEASON");
        return bean;
    }
}
```

Update Business Object or Value Object

To customize the attributes on a business/value object within the system appropriately, subclass the business/value object in question and add the attribute to the sub-class only:

```
public class GriItem extends Item {
    private String season;
    public GriItem(String itemId) {
        super(itemId);
    }
    public String getSeason() {
        return season;
    }
    public void setSeason(String season) {
         this.season = season;
    }
}
```

Next, subclass the BOFactoryImpl to override the item creation method and replace it with your own. Once configuration is altered to use the custom factory, everywhere in the code where an Item would have been created/instantiated, a GriItem is instantiated instead:

```
public class GriBOFactoryImpl extends BOFactoryImpl {

    public Item createItem(String itemId) {
        return new GriItem(itemId);
    }
}
```

To configure the item, simply modify the common.cfg file with the classpath of your new code:

```
BO_FACTORY_IMPL=gri.custom.code.GriBOFactoryImpl
```

Update the Services

The same process used to customize the objects within the system is used to override services at the service layer. A subclass of the service is made, followed by pointing the correct configuration file at the subclass instead of the original SIM service:

```
public class GriItemServerServices extends ItemServerServices {
  public StockItem readStockItem(String itemId, Long storeId) throws Exception {
     StockItem stockItem = super.readStockItem(itemId, storeId);
     GriItem griItem = (GriItem) stockItem.getItem();
     griItem.setSeason(new GriItemAttributeDao().selectSeason(itemId));
     return stockItem;
  }
}
```

When overriding an implementation method in the service, it is advisable to call the superclass for standard SIM processing and then perform customized processing afterwards. This ensures continued functionality with future releases.

After creating the customized service, create a customized factory as with the business objects (extending SIM's ServerServiceFactoryImpl), and then configure the common.cfg file to use your customized factory implementation:

```
public class GriServerServiceFactoryImpl extends ServerServiceFactoryImpl {
    public ItemServices createItemServices() {
        try {
            return new GriItemServerServices();
        } catch (Throwable t) {
            LogService.error(this, "Could not create GriItemServerServices", t);
            return null;
        }
    }
}
```

Example 7–1 common.cfg

```
SERVER_SERVICE_FACTORY_IMPL=gri.custom.code.GriServerServiceFactoryImpl
```

Service Index

The following is a current list of services within SIM:

- ActivityLockingServices
- AdhocCountAdminServices
- AuditLogServices
- BatchServices
- BillOfLadingServices
- ConfigServices
- CustomThemeServices
- CustomerOrderServices
- DealServices
- InventoryAdjustmentServices
- ItemServices
- ItemBasketServices
- ItemRequestServices
- ItemTicketServices
- MdseHierarchyServices
- PollingTimerAdminServices
- PosTransactionServices
- PriceChangeServices
- ProductGroupServices
- ProductGroupScheduleServices
- ReplenishmentServices

- ReportFormatServices
- ReportingServices
- ReturnServices
- SaleReturnServices
- SecurityServices
- SequencingServices
- ShipmentServices
- SourceServices
- StockCountServices
- StockCountLineItemServices
- StockCountLocationServices
- StoreServices
- StoreOrderServices
- TransferServices
- TranslationServices
- UINServices

Update the PC Screen

In order to view or enter additional information on the PC, a new UI space must be created instead of modifying current screens. This is done by adding an additional button to the navigation of the application. In order to add a new button, a new row must be put into the PC_MENU_ITEM table.

Table 7–1 PC_MENU_ITEM Table

Column	Value to Place in Column
MENU	This contains the fully qualified classpath to the screen that contains the menu.
NAME	This is the name of the button to be displayed on the menu. This value is used as a translation key by the client to get the displayable text.
PERMISSION	This is the permission key associated to the button. If the user does not have the assigned permission, the button does not display. **Null** is valid.
MENU_ITEM_ACTION	This can be one of three values: NONE, BACK or another menu.
MENU_ITEM_ORDER	Buttons are displayed for the MENU in the sequence of lowest to highest MENU_ITEM_ORDER value.
IS_DEFAULT	**Y** if the button should be the default button for the window. **N** if not the default button for the window.

If **NONE** is entered in MENU_ITEM_ACTION, then when the navigation button is clicked, no navigation takes place. If BACK is entered, then when the navigation button is clicked, the screen navigates to the previous screen in the workflow. If a value matching a MENU column in the table is entered (another menu/screen), then when the navigation button is clicked, the application navigates to the specified screen. In all three of these scenarios, the button action is first sent to the actionPerformed() method of the screen before any navigation takes place.

An example record to add a button to the item detail screen looks like the following:

Table 7–2 PC_MENU_ITEM Table, Item Detail

Column	Value
MENU	oracle.retail.sim.shared.swing.item.ItemDetailScreen
NAME	Additional Details
PERMISSION	null
MENU_ITEM_ACTION	gri.retail.sim.custom.AdditionalItemDetailScreen
MENU_ITEM_ORDER	999
IS_DEFAULT	N

The following is an example record to create a menu for the new custom screen:

Table 7–3 PC_MENU_ITEM Table, Custom Screen

Column	Value
MENU	gri.retail.sim.custom.AdditionalItemDetailScreen
NAME	Cancel
PERMISSION	null
MENU_ITEM_ACTION	BACK
MENU_ITEM_ORDER	1
IS_DEFAULT	N

Next, create the AdditionalItemDetailScreen class which must extend SimScreen and implement all the required methods. After that, design and build the screen (see Adding New Workflows).

Update the HH Forms

Customizing the HH application is more difficult than other areas of the application. Wavelink does not supply simple or straightforward ways to customize their forms. In order to add additional item information to the HH item lookup workflow, the handheld SIM code source files need to be modified directly.

It is strongly suggested that all new features, information or processes be added with completely new handheld forms. For how to build forms in the Wavelink server, see the Wavelink documentation. Once the new forms are built, a single form may be customized to contain a menu option or command to navigate to the new workflow. This will minimize the amount of rework that needs to be done with each release update of the handheld source code.

Adding New Workflows

Rather than just adding functionality to an existing workflow, this section describes how to add a new workflow to SIM. It focuses on how to design and implement the new workflow to minimize the impact of upgrades to the customization. The example workflow added to the system is the ability to create new serial numbers in an In Stock status without moving them using a system transaction, that is, entering serial numbers into the UIN_DETAIL table without receiving them or creating an inventory adjustment. This type of functionality is used primarily to data seed serial numbers that should have already been in the system.

The following topics will be covered:

- Building Business Objects
- Building Enterprise Java Bean (EJB) Services
- Building Data Access
- Building PC Screens
- Building Wireless Forms
- Exceptions

Example Code

This section references example code that is packaged independent of this document. For more information about this example code, see Appendix: Code Examples.

Table 7–4 lists the various files and their general description.

Table 7–4 Example Code Files

Name	Description
CustomSerialNumber	Business object representing the new serial number.
CustomUINBean	The Enterprise Java Bean (EJB) that implements the CustomUINInterface.
CustomUINCountTableEditor	Table editor used in the UI CustomUINCreateScreen.
CustomUINCreateDialog	Popup dialog that enables users to enter the new UINs.
CustomUINCreateDialogModel	Model for the popup dialog.
CustomUINCreateDialogTableEditor	Table editor used in CustomUINCreateDialog to enter individual UINs.
CustomUINCreateModel	Model for the create screen (track data, talk to server).
CustomUINCreatePanel	Panel for the create screen (contains UI widgets).
CustomUINCreateScreen	Screen for the create screen (interfaces with navigation system).
CustomUINCreateWrapper	A client-side wrapper object for the custom serial number.
CustomUINEJBServices	Client-side EJB Service that looks up and accesses the EJB bean.
CustomUINInterface	Defines the API for Custom UIN EJB.
CustomUINOracleDAO	Database Access Object for custom UIN create logic.

Table 7–4 (Cont.) Example Code Files

Name	Description
CustomUINServerServices	Implementation of CustomUINServices.
CustomUINServices	Defines the API for Custom UIN Services.
CustomUINDataBean	A data bean for a new theoretical CUSTOM_UIN table.

Building Business Objects

In the SIM architecture, business objects represent basic concepts within the Store Inventory Management domain. Some examples of significant objects within SIM are Item, Store, StockCount, Transfer, Shipment and Receipt. Most business objects contain very little business logic. Rather, a business object contains the data associated with the domain concept. The majority of code within a business object concerns itself with getting and setting data values on the business object. The business logic associated with the concept is contained within the ServerServices (or service layer) that uses the business object. Because they contain all the data, business objects flow across all three layers of the SIM architecture.

These business data objects can be designed in any manner as long as they implement Serializable and Cloneable so that they can be used as parameters to server APIs. If the business object extends SIM's BusinessObject class, this is automatically done as well as gaining other useful methods to use.

See CustomSerialNumber.

The Business Object Class

All SIM business objects extend from the single class BusinessObject. Many classes can be in a hierarchy chain, but BusinessObject should always be the top level object. Features provided by the class include:

Table 7–5 BusinessObject Features

Name	Description
isAttributesEqual()	Return **true** if two values are equal.
isAttributesNotEqual()	Return **true** if two values are not equal.
isPropertyModifiable()	Return **true** if the property specified is modifiable; **false** otherwise. This method makes an executeRule() call, but wraps the call in a try/catch block. If the rule returns an exception, the block catches the exception and returns **false**, otherwise it returns **true**. The implementation of the rule is just like that of any other business rule. The rule makes whatever checks are necessary to determine if the attribute is modifiable. It returns a RulesInfo object containing a text message if it is not modifiable, otherwise is returns an empty RulesInfo object. To alter this method, either override the method in business object sub-class (for example, CustomSerialNumber.java) or configure a rule for the object to execute.
isCoherent()	Returns **true** if the object is currently coherent. This method makes an executeRule() call. It throws a business exception if the object is not coherent and returns **true** if it is.
executeRule()	Methods that access the rules engine. This allows business rules to be externally defined against business objects in a configuration file. When a method is called, the rule engine might dynamically load and execute business rules defined for a method.
checkForNullParameter()	Method to determine if a parameter is null and throws a business style exception if it is.

Table 7–5 (Cont.) BusinessObject Features

Name	Description
clone()	A default implementation of clone() is provided. This implementation provides a deep copy of the object. All objects referenced by the business object are also fully copied. Often, this might not be what the developer wants.
cloneShallow()	Clone method that is the same as the standard Java object implementation of clone().
toString()	Uses reflection to convert all attributes of the BusinessObject to a single string.

There are two more types of data objects used within SIM to communicate between the client and server: the query filter and value object.

Query Filter

Business objects represent a single concept of the domain. Sometimes the objects must be selected in groups, often by some criteria, such as finding all employees whose last name begins with **T**. For those services that require filter criteria, a QueryFilter business object is created. These filters are used in the DAO layer to construct WHERE clauses when selecting and populating the business objects. There is no difference between creating a business object and creating a query filter object (which is also a business object):

```
public class MyQueryFilter extends BusinessObject implements QueryFilter
```

Value Object

A value object (VO) is a trimmed-down version of a business object that contains only the attributes needed for a very specific piece of the application. These objects are created to reduce data flow in areas that do not require that the attributes be updated or any business logic:

```
public class MyVO implements Serializable
```

There is no specific benefit or advantages to creating a value object or query filter other than consistency with naming patterns in SIM while doing customizations.

Customization Using the Rules Engine

The set() methods on business objects and query filters (but not value objects) always call executeRule(). When a method is called, the rule engine might dynamically load and execute business rules defined for a method. This enables business rules to be externally defined against business objects in a configuration file.

Rules are simple classes that validate business logic upon various objects within the system. They are executed primarily when attributes are set() on business objects, but there are also rules accessed when isPropertyModifiable() or isCoherent() are executed. There are already many rules defined in the system. Before creating a new rule, look through existing rules to see if the one you need already exists.

The business rules are located in the numerous *.rules.* packages. If a desired rule does not exist, do the following to create a new rule.

Creating a New Business Rule

To create a new business rule, create a new java class that extends SimRule. This enables the rule to interact with SIM's rule framework. QuantityCannotBeNegativeRule is being used as an example of writing a business rule. This rule passes if the item is sellable and fails if it is not.

```
public final class QuantityCannotBeNegativeRule extends SimRul
```

Override the execute() method. This first method performs some brief validation of the args parameters, but the standard coding practice is to break out the args array and cast to specific types to be passed to a second execute() method that performs the logic of the business rule. In the following example, the object parameter passed in by the rule engine is not needed. The first parameter of the array is the Quantity that needs to be validated:

```
public RulesInfo execute(Object object, Object[] args) {
     return execute((Quantity) args[0]);
}
```

Implement the typed execute() method with the actual logic necessary to perform the desired validation:

```
private static final RulesInfo RULE_FAILED
              = new RulesInfo(ErrorKey.QUANTITY_INVALID_NEGATIVE);
private RulesInfo execute(Quantity quantity) {
    if ((quantity != null) && (quantity.doubleValue() < 0)) {
        return RULE_FAILED;
    }
    return RULE_PASSED;
}
```

In the previous example, RULE_PASSED is returned at the end of the validation to indicate that no failure took place. Note that RULE_PASSED is actually an empty RulesInfo object declared in the SimRule super-class that should be used in all subclasses at the appropriate spots.

A RulesInfo wrapped around an error string indicates that the rule failed. The framework handles this failed rule information and converts it into the appropriate exception.

Logic in rules is intended strictly for validation checking. The logic should never update or modify the actual object that it is validating. Doing this would violate the contract of the rules engine in SIM and likely leave the business object in a non-coherent state.

Once the rule is developed, the rules configuration file must be updated. Update all occurrences of the rules_sim.xml file:

Example 7–2 rules_sim.xml

```
<object className="oracle.retail.sim.closed.invadjustment.InventoryAdjustment">
    <property id="setPackSize">
        <rule_class className=
           "oracle.retail.sim.closed.rules.common.PackSizeMustBePositiveRule"/>
    </property>
    <property id="setQuantity">
        <rule_class className=
           "oracle.retail.sim.closed.rules.common.QuantityCannotBeNegativeRule"/>
    </property>
    <property id="setComments" propertyNames="setComments">
```

```
        <rule_class className=
           "oracle.retail.sim.closed.rules.common.MaxCommentSizeRule"/>
    </property>
</object>
```

This xml file contains a list of classes and rules to execute when certain properties are modified. At the top level, an object is defined with a className containing the complete path to the object on which the validation should be done:

```
<object className="oracle.retail.sim.closed.invadjustment.InventoryAdjustment">
</object>
```

Within the class definition is a property definition where the ID is the method signature on which the validation should be performed:

```
    <property id="setQuantity">
    </property>
```

Within the property definition is the rules class definition where className is assigned the fully qualified path to the rule:

```
<rule_class className=
           "oracle.retail.sim.closed.rules.common.QuantityCannotBeNegativeRule"/>
```

Building Enterprise Java Bean (EJB) Services

The next step in creating the workflow is defining the server API that is accessed to accomplish the work. This means designing an EJB to deploy in the server:

1. Define the Service API (see CustomUINServices).

   Define the exact API to call on the server to perform some logic. These services files might have numerous APIs associated with them.

2. Define the API in an EJB interface (see CustomUINInterface).

   The parameters and return value must be defined as CompressedObject if the API wants to take advantage of SIM's compression during communication between the client and server. A CompressedObject return value should be defined even if the return value is **null** or was defined in step 1 as **null**.

3. Develop the EJB Bean (see CustomUINBean).

   When the EJB bean is developed, extending AbstractSimServiceBean enables quick access to helper methods such as handleException(), getSessionContext() and completeServiceContext().

   The EJB Bean must implement the interface defined in Step 2. In addition, note that to use the compression built in to SIM for quicker performance, even if the service is defined as null, a CompressedObject must be returned. Note that using the try/catch/finally allows the smooth handling of all exceptions and the completion of the service context even if the EJB failed with an exception.

4. Develop EJB Services (sse CustomUINEJBServices).

   The EJB Services class is designed to be the client-side of a service call. It handles the task of looking up the EJB Bean, instantiating the compressed objects around the arguments and making the call to the EJB Bean. The EJB Services use the EJBServicesManager to do cached lookups of the EJB Beans for improved performances. Following the pattern in the example is the easiest way to create an EJB Service.

5. Develop EJB Server Service (see CustomUINServerServices).

   The EJB Server Services class contains the actual server side code that performs the business logic of the API. In this particular example, the code only needs to save the new serial numbers, so a DAO (data access object) is instantiated and its API accessed.

6. Deploy EJB Services.

   The new EJB Bean needs to be deployed to the server. A system administrator or lead developer should be able to handle this process. The steps are normal steps for deploying an EJB into a server:

   a. The EJB must be packaged in a standard ejb style .jar.

   b. A reference to that .jar must be include in the META-INF/application.xml within the sim-server.ear.

Building Data Access

The next step is to write the DAO that communicates with the database. Using some of SIM's pre-existing tools can simplify this task.

Database Layer Development Tips

- Removing a column from the database breaks the code that attempts to read that table, so do not remove columns.
- New tables should always begin a with a custom client-created prefix to keep it easily distinguishable from basic SIM tables. If the company name is Business Company Inc., then a table containing additional item information might be called BCI_ITEM_INFO.
- New columns added to an existing database table should begin with a custom prefix. If a company named Business Company Inc. wanted to add a new column to captured information about a stock count to the stock count header table, it might look like STOCK_COUNT.BCI_EXTRA_DATA.
- All new columns added to the existing tables must be nullable.
- If there is a foreign key reference in newly added tables, then keep the same column name as the table column name it references.
- Do not modify any of SIM's DAOs. Either sub-class and configure the DAO or create a completely new custom DAO.
- Ensure the DAO sub-classes BaseOralceDao so that connection pooling is handled properly.
- Ensure that all data beans sub-class FullDataBean or BaseDataBean.

Creating a DAO

Refer to Update DAO (Database Access Objects) for information about developing new DAOs. Also, refer to CustomUINOracleDAO for a code example.

The remainder of this section contains suggestions for using general SIM DAO classes:

Table 7–6 DAO Layer Framework Classes

Name	Description
BaseOracleDao	Abstract base class for all Oracle data access objects. Every DAO implementation must be a sub-class of this object. This class provides a range of generic functionality as well as access to the database connection factory.
BaseDataBean	Abstract base class for all data access beans. Every database bean should have this in its hierarchy.
BatchParametricStatement	Encapsulates SQL and the parameters to be used during its execution. It creates a batch of statements and executes them at the end. This performs much better under the scenario where the same SQL statement is executed multiple times.
FullDataBean	Abstract base class for all data beans that contain select, insert, update and delete functionality.
ParametricStatement	Encapsulates the SQL and the parameters used during the execution of the SQL.
SimStoredProcedure	Interface for any java code related to stored procedures. Classes implement this interface and are passed to the BaseOracleDao.

DAO Methods

Almost all DAO methods consist of creating a ParametricStatement or BatchParametricStatement and then calling the appropriate execute() method in the BaseOracleDao. In the saveSerialNumbers() method of the custom UIN example, a custom batch SQL statement is built using the UINDetailDataBean class attributes. Regular inserts and updates of a table are much simpler. For example, if a table called CUSTOM_UIN was created to temporarily hold our input, some simpler examples of how the DAO layer works (see CustomUinDataBean.java) might be available.

The following would be the save method in the dao layer using the new data bean:

```
public void saveSerialNumbers(Long storeId, List<CustomSerialNumber>
serialNumbers)
                                                                    throws
SimServerException {
    String sql = CustomUinDataBean.UPDATE_SQL
                      + where(CustomUinDataBean.COL_ID, CustomUinDataBean.COL_
STATUS);
    BatchParametricStatement batchStatement = new BatchParametricStatement(sql);
    CustomUinDataBean bean = new CustomUinDataBean();

    for (CustomSerialNumber serialNumber : serialNumbers) {
        bean.setId(serialNumber.getId());
        bean.setStatus(UINStatus.IN_STOCK.getCode());
        bean.setStoreId(storeId);

        List<Object> params = bean.toList(false);
        params.add(serialNumber.getId());
        params.add(serialNumber.getStatus().getCode());

        batchStatement.addParams(params);
    }
    executeBatch(batchStatement);
}
```

Building PC Screens

The SIM PC client is a Swing application. It is launched using WebStart and a browser. It communicates with the SIM server through EJBs. The application does not support offline functionality. If communication with the server is lost, an error message is displayed and the client is returned to its login screen.

> **Note:** Because the EJBs are services that run on the server, any client may be written against these remote services. A thin-client application could be written to take advantage of these services if the client wanted some functionality available through web pages.

Customization Guidelines for the PC

- Always access the EJB services through the ClientServiceFactory class.
- Do not modify any framework-related code.
- To avoid conflicts with updates, only add functionality to client through adding navigation buttons that navigate to custom screens.

PC Client Architecture

The client architecture is broken into five layers, only three of which are worked on to create new or customized PC functionality:

- Application Framework
- <name>Screen
- <name>Panel
- <name>Model
- EJB Service

Application Framework

There is an extensive framework of Swing-related classes used to launch and control the application as well as tools to make working within the Swing client easier. This framework is located in oracle.retail.sim.closed.swing.* packages. There is a wide area of functionality covered in this framework from layout tools, to customization tools, to internationalization hooks, to a navigation engine, to advanced tables and advanced widgets. These framework classes should not be modified.

<name>Screen

A screen is the top level of a single PC screen display. Its responsibility is to interact with the navigation system and delegate the actions of its menu items to the panel (see CustomUINCreatePanel).

<name>Panel

The panel is where all the visual elements of a single PC screen are located. All the Swing widgets are declared here as well as any functional logic that touches those widgets (getting and setting the widget properties and values). All logic not associated directly with the widgets is delegated to the model (see CustomUINCreatePanel).

<name>Model

The model is where all access to the service layer takes place as well as any complex logic that needs to take place on the client is written (see CustomUINCreateModel).

EJB Service

An EJB service is accessed through the ClientServiceFactory (or custom written EJB service) and should only be done from the application framework or from the model layer.

Navigation

There is both configurable navigation and hard-coded navigation within the SIM application.

External Configurable Navigation The primary means of navigation in the PC application is through clicking the menu buttons displayed at the top of the screen. These menu buttons are determined using the information in PC_MENU_ITEM in the database. Adding, editing, and removing buttons for customization must be done within by adding or altering information in this database table. See Update the PC Screen for more information.

Hard-Coded Navigation The ability to navigate from within the <name>Screen or <name>Panel class is supplied in the framework by the navigate() and navigateLater() methods found in SimScreen and ScreenPanel classes. There are two different ways that hard-coded navigation takes place. The DirectDeliveryDetailScreen/Panel is an excellent example of this. Because it can be reached through screens that should not be returned to using the standard framework, very specific navigation is coded for the screen.

For example, in the handleDelete() method, the application return to the previous screen only if the direct delivery is successfully cancelled. This is done using the SimNavigation.BACK value as the navigate parameter. The remainder of the values in SimNavigation are the identities of the buttons used within the application:

```
public void handleDelete() throws Exception {
    lineItemTable.stopEditing();
    // Rest of method code removed
    if (model.isDirectDeliveryEmpty() && cancelEmptyDelivery()) {
         navigate(SimNavigation.BACK);
    }
}
```

Screens

A screen is the top level of a single PC screen display. The primary responsibility of screens is to interact with the navigation framework. The following is a break-down of the structure of screens. Use these coding guidelines when creating a customized screen. The example is the new CustomUINCreateScreen built to add the customized functionality of creating UINs without going through a transaction.

1. The UI screen must extend SimScreen. Usually, the only declaration is the panel that the screen delegates to. In some rare cases, variables may need to be declared to track some value at the screen level. Declare the constructor, which never contains parameters and simply adds the panel using the add() method available on the superclass.

   Implement the getScreenName() method. This should return the title of the screen to be displayed.

   ```
   public class CustomUINCreateScreen extends SimScreen {
         private static final long serialVersionUID = 487712289865375658L;
         private CustomUINCreatePanel panel = new CustomUINCreatePanel();
         public CustomUINCreateScreen() {
             add(panel);
   ```

```
    }
     public String getScreenName() {
         return "UIN Create";
    }
}
```

2. Implement and start() method. This method is called whenever the screen is navigated to. The start() method is called as the navigation takes place. Of interest are the two lines of code that are found in nearly every start() method. showMenu() causes the menu of the screen to be displayed. Passing in a parameter assigns a default button for the screen. The call to panel.start() triggers the startup of the visual panel. There can be a wide variety of functionality necessary in the startup of a screen, but it is important to remember that all functionality should be delegated to the panel except for what is related to navigation. The start() method may throw an exception which will halt the navigation to the screen.

```
public void start() throws Throwable {
        showMenu();
        panel.start();
}
```

3. Implement the stop() method. This method is called whenever the screen is exited. Place code in here to clean up the state of the screen. This method can never throw an exception.

```
public void stop() {
        panel.stop();
}
```

4. Implement the resume() method if necessary. Resume is executed when the screen is navigated away from, but has not disappeared from the chain of screen. When the screen is returned to (such as from a sub-screen, this method is called to resume the screen). The resume() method should always call showMenu() to reset which buttons should be visible. The super-class SimScreen has a default implementation:

```
public void resume() {
    showMenu();
}
```

5. There are two new methods:

   - isStartable()
   - isStoppable()

   They are checked before start() and stop() are called respectively when screen navigation is taking place. Screen state validation code can be placed in these methods. If isStartable() returns **false**, the screen will not allow itself to be navigated to. If isStoppable() returns **false**, the screen will not allow itself to be navigated away from:

```
public boolean isStartable() {
        return panel.isStartable();
}
```

6. Override the navigationEvent() method from the superclass. All menu buttons create NavigationEvent objects and pass them to the screen through this method. Each event has a command value that matches the button that clicked it. The implementation of this method should determine which button is clicked and delegate to the appropriate panel method:

```
public void navigationEvent(NavigationEvent event) {
       String command = event.getCommand();
       try {
            if (command.equals(SimNavigation.ADD_ITEM)) {
                panel.doAddItem();
            } else if (command.equals(SimNavigation.DELETE_ITEM)) {
                panel.doDeleteItem();
            } else if (command.equals(SimNavigation.DONE)) {
                panel.doHandleDone();
            }
       } catch (Throwable exception) {
            displayException(panel, event, exception);
       }
}
```

Features of the Screen The SimScreen superclass contains several features that can be used by all subclasses. A brief listing is supplied here.

Table 7–7 Features of the Screen

Feature	Description
assignFocusInScreen()	Assigns focus to the first button on the navigation menu.
navigate()	Navigates to the specified screen.
navigateLater()	Navigates to the specified screen on a separate thread.
removeFromScreenHistory()	Removes the screen from the navigational history.
removeNavButton()	Removes a navigational button from the menu.
displayException()	Displays the exception in the error dialog.

Screen Panels

The panel is where all the visual elements of a single PC screen are located. All the Swing widgets are declared here as well as any functional logic that touches those widgets (getting and setting the widget properties and values). All logic not associated directly with the widgets is delegated to the model. The following shows some of the basic principles of designing a panel (see CustomUINCreatePanel):

1. Declare the panel. The new panel should extend the ScreenPanel class.

```
public class CustomUINCreatePanel extends ScreenPanel implements REventListener
{
    private CustomUINCreateModel model = new CustomUINCreateModel();
```

2. Declare Editors and Tables. Declare the editors and tables that will be used within the panel. There should be an editor for each style of data currently used in SIM. See Editors for further detail. SimTable is the most common type of table used in SIM. SimTable should be placed within a SIMTablePane.

```
private StockItemTableEditor stockItemTableEditor = new StockItemTableEditor();
private SimTable stockItemTable = new SimTable(new
CustomCreateUINTableDefinition());
private SimTablePane stockItemPane = new SimTablePane(stockItemTable);
```

3. Declare Constructor. The constructor of a panel is expected to not take any parameters and not throw any exceptions:

```
public CustomUINCreatePanel() {
    initializePanel();
    layoutPanel();
}
```

4. Initialize the Panel. The following is an example of initializing the property settings on editors and tables:

```
private void initializePanel() {
     stockItemTableEditor.addTableEditorListener(buildStockItemListener());
     stockItemTable.setColumnSize("serialNumberCount", SimTable.LABEL_WIDTH);
}
private void layoutPanel() {
    setContentPane(stockItemPane);
}
```

5. Start the Panel. A start() method should be created so that the screen can call it to load the panel with information. Note that if a default focus editor is desired, the assignFocusInScreen() method should be called at the end of start passing in the component to receive focus.

```
public void start() throws Throwable {
     stockItemTable.addRow(new CustomUINCreateWrapper());
}
```

Editors

The following is a brief list of common UI data editors used in SIM and what they do.

Table 7–8 Editors

Editors	Description
RCheckBoxEditor	A check box that can be selected or de-selected.
RComboBoxEditor	A combo box list of selections.
RDateFieldEditor	A single date field that can be edited by hand or through a calendar.
RDateRangeEditor	Two date fields that represent a start date and end date.
RDecimalFieldEditor	A text field that only allows decimal numbers to be entered.
RDisplayLabelEditor	An object displayer that does not allow editing.
RIntegerFieldEditor	A text field that only allows integer numbers to be entered.
RListEditor	A scrollable list of selections.
RLongFieldEditor	A text field that only allows long numbers to be entered.
RLongTextFieldEditor	A text field with an expansion dialog associated to it, often used for very long text fields that must take up limited screen space.
RMoneyFieldEditor	A text field that only allows money values to be entered.
RNumericIdEditor	A text field that edits numeric-only identifier strings.
RPasswordFieldEditor	A text field that allows the hidden entry of a password.
RPercentFieldEditor	A text field that only allows percent values to be entered.
RQuantityEditor	A text field that only edits Quantity values.

Table 7–8 (Cont.) Editors

Editors	Description
RRadioButtonEditor	An editor that allows a set of option buttons to be displayed and selected.
RSearchComboEditor	An editor that present a combo box of values but allows the user to trigger a search for additional values.
RSearchFieldEditor	A text field that allows the user to enter a value or search for one.
RTextAreaEditor	A large text area where large quantities of text can be edited.
RTextFieldEditor	A single line text field where small quantities of text can be edited.

Initializing Editors The editors are customized to contain numerous properties beyond those of the normal Swing widgets. Many of the customized properties are just short-cuts to other work. Here is a sample of some initialization in the InventoryAdjustmentFilterDialog. This includes setting displayers (which are responsible for formatting data), setting general size properties for the editors, and assigning an identifier to a text entry field (without an identifier, the editor will not activate).

Example 7–3 InventoryAdjustmentFilterDialog

```
private void initContent() {
   adjustmentEditor.setIdentifier(SimName.INVENTORY_ADJUSTMENT_NUMBER);
   adjustmentEditor.setEntryAlignmentLeft();
   adjustmentEditor.setSizeType(EditorConstants.MEDIUM);
   statusEditor.setDisplayer(new TranslatedObjectDisplayer());
   statusEditor.setSizeType(EditorConstants.LARGE);
   statusEditor.setEmptyType(RComboBoxEmptyType.ALL);
}
```

Editor Layout To lay out editors in a simple grid, use an REditorPanel. Declare the rows and columns when instantiating and add editors to the panel. The REditorPanel handles all the layout details:

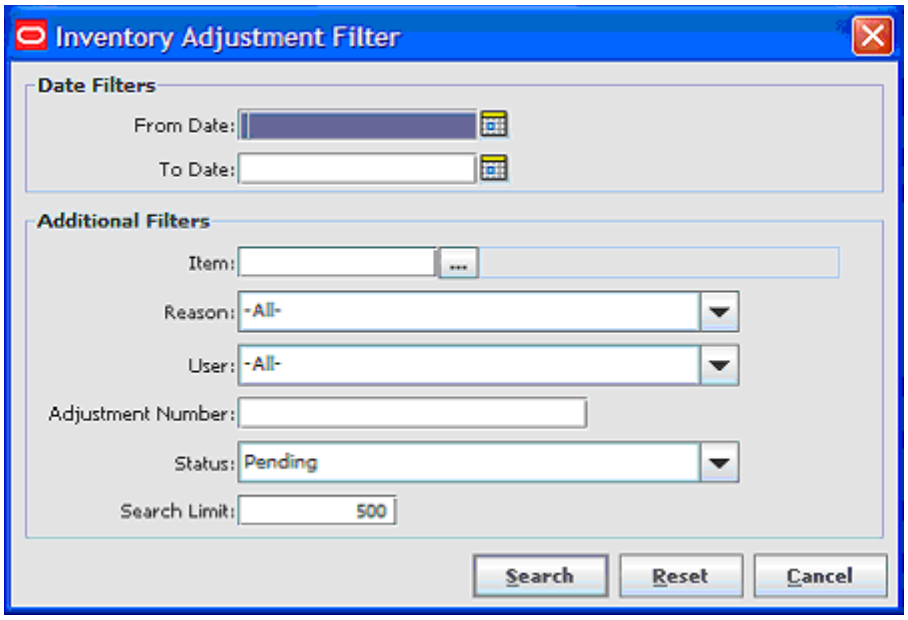
```
private void layoutContent() {
    REditorPanel miscFilterPanel = new REditorPanel(6);
    miscFilterPanel.setTitleBorder("Additional Filters");
    miscFilterPanel.add(itemEditor);
    miscFilterPanel.add(reasonEditor);
    miscFilterPanel.add(userEditor);
    miscFilterPanel.add(adjustmentEditor);
    miscFilterPanel.add(statusEditor);
    miscFilterPanel.add(searchLimitEditor);
}
```

Search Editor

A search editor is an editor that enables a user to enter data or find data. It consists of an editable text field that allows the user to enter an ID, a button that allows the user to search, and a non-editable text field that displays the description. The InventoryAdjustmentFilterDialog has a search editor for its item value that will be used as an example.

If an ID is entered, the editor automatically searches for the whole object. When the button is clicked, the screen navigates to a dialog where an object is chosen. Upon return, the data of the object is displayed:

Figure 7–1 InventoryAdjustmentFilterDialog



Using a Search Editor Do the following to declare and implement a search editor. Examples are from InventoryAdjustmentFilterDialog.

1. Declare the search editor.

```
private RSearchFieldEditor itemEditor =
SimEditorFactory.createItemVOSearchFieldEditor(false);
```

2. Set all the properties of the search editor. The following basic properties are required for a search editor to function:

Table 7–9 Search Editor Properties

Property	Description
identifier	The identifier is a component's name and can be used to reference the component uniquely.
search processor	The search processor is a class that implements the SearchProcessor interface. When the ID is altered in the entry field, this processor will be triggered to attempt to find the information.
search listener	The search listener is a class that implements the SearchListener interface. This class will be called when **Search** is clicked.

Example of setting up the properties of the search editor:

```
itemEditor.setIdentifier(SimName.ITEM_ID);
itemEditor.setSearchProcessor(new
ItemVOSearchProcessor(allowNonInventoryItems));
itemEditor.setSearchListener(buildItemSearchListener());
```

3. Build a search listener. A search listener needs to be declared independently for each search editor that is used. This class must implement the SearchListener interface. Basically, it contains the method that is called once the data is found.

```
private ItemSearchListener buildItemSearchListener() {
    return new ItemSearchListener() {
        public void assignItem(ItemVO itemVO) {
            if (itemVO != null) {
                itemEditor.setData(itemVO);
            }
        }
    };
}
```

4. Build a search processor. A new search processor must implement the SearchProcessor interface. It must return a display for the entry area of the widget, a displayer for the value area of the widget, it must implement the searchById() method to find the appropriate object can put in a method that validates the data.

```
public class UserSearchProcessor implements SearchProcessor {
  private AttributeDisplayer entryDisplayer = new
AttributeDisplayer("userName");
  private DualAttributeDisplayer valueDisplayer
              = new DualAttributeDisplayer("firstName", "lastName",
StringConstants.SPACE);

public Object searchById(String userName) throws Exception {
    User user = ClientServiceFactory.getSecurityServices().readUser(userName);
    if (user == null) {
        UIException exception = new UIException(ErrorKey.USER_NOT_FOUND);
        exception.addValue(userName);
        throw exception;
    }
    return user;
  }
  public BasicDisplayer getEntryDisplayer() {
      return entryDisplayer;
  }
  public BasicDisplayer getValueDisplayer() {
      return valueDisplayer;
  }
  public Object validateData(Object data) {
      return data;
  }
}
```

SimTable

SimTable is an extension of JTable that allows the display and editing of cells and data. It contains advanced options not available with the standard table. It is also specifically designed to handle common design patterns in SIM.

Using a SimTable Declare the SimTable globally within the panel. It requires passing in a table definition. The SimTable should be placed within a SimTablePane and the pane added to the layout. SimTablePane supplies the scrollbars for the table:

Example 7–4 CustomUINCreatePanel

```
private SimTable stockItemTable = new SimTable(new
CustomCreateUINTableDefinition());
private SimTablePane stockItemPane = new SimTablePane(stockItemTable);
```

When initializing the screen panel, set desired properties on the table. In the example workflow code, we limit the size of the serial number count column to the width of its label. In the ItemLookupPanel code we show an example of other simple features of the table being applied.

Example 7–5 CustomUINCreatePanel

```
stockItemTable.setColumnSize("serialNumberCount", SimTable.LABEL_WIDTH);
```

Example 7–6 ItemLookupPanel

```
itemTable.setTableEditable(false);
itemTable.setSingleRowSelectionMode();
itemTable.registerDoubleClickAction(this, ITEM_SELECTED);
```

Assigning data to the table is simply a matter of passing a collection of objects to be displayed. The setRows() method automatically replaces all the content of the table. The objects must all be of the class type declared in the table definition. There are other common methods for dealing with data such as addRow(), insertRow(), updateRow() and removeRow() that simply take the data object to be displayed in the row.

Example 7–7 ItemLookupPanel

```
itemTable.setRows(model.findItemVOs(searchFilter));
```

Example 7–8 CustomUINCreatePanel

```
stockItemTable.addRow(new CustomUINCreateWrapper());
```

Retrieving information from the table is as simple as using the various available get() methods such as getSelectedRow(), getSelectedRowData(), getAllSelectedRowData(), and getAllRowData(). Methods ending in Data retrieve the object represented by the row, so getSelectedRow() returns the row index, whereas getSelectedRowData() returns the row object. For example, see the doDeleteItem() and doHandleDone() methods in CustomUINCreatePanel.

SimTableDefinition A table definition is the core of how a table works. To create a table definition, develop a class (public or inner private) that extends SimTableDefinition, and then implement the required methods. See CustomCreateUINTableDefinition in CustomUINCreatePanel as an example of this class. The following is a quick summary of APIs on SimTableDefinition class.

Table 7–10 SimTableDefinition APIs

API	Description
getIdentifier()	The unique identifier of the table. This API is used to track configuration information. The default implement is declaringClassName.className.
getSortAttributes()	Retrieves the list of default sort attributes. If the table configuration does not have any sort configuration settings, these sort attributes will be used to generate the initial sort configuration.
getOverrideEditableAttributes()	Retrieves a list of attributes that do not require having a setter() method on the model in order to be considered editable in a table.
getDataClass()	The Class object of the data that each row in the table represents. This operates as the model for a row.
getAttributes()	A list of SIM table attributes that represent the attributes of each column.

In some cases, a single row of the table actually represents data that is originally from two different business objects. In the case of a table where the column definitions do not match with a data class one-for-one, a wrapper class should be created to wrap the data objects. The wrapper will contain the actual data, but have method APIs that represent the columns in the table.

SimTableAttribute A SimTableAttribute represents exactly one column within the SimTableDefinition. There are many different properties that can be assigned to the object. See CustomUINCreatePanel for the example of CustomCreateUINTableDefinition:

Title
The title is displayed as the column header. This text label is actually a key into the translation tables. This key is also used as a unique identifier for the column.

Attribute
The second parameter is the attribute of the column. The value represents a getter and setter on the class type for retrieving the data. For example, if shortDescription is the attribute, then getShortDescription() and setShortDescription() should exist on the class type defined in the getDataClass() method. The attribute should always begin with a lower case letter. If a period appears within the attribute, then multiple levels of method calls with take place. For example, the attribute one.two.three would be converted into getOne().getTwo().getThree() when attempting to retrieve the column information from the data class. However, layered method calls like this are a good indicator that a wrapper or value object needs to be created.

Displayer
Each attribute value is examined by the table during instantiation and default displayers are assigned for the data type belonging to the attribute. If the attribute is a Quantity, then a QuantityDisplayer is assigned; if the attribute is a Boolean, then a BooleanDisplayer is assigned, and so on. If the attribute requires specialized formatting, then a displayer should be assigned to the attribute manually. In the example, an AttributeDisplayer is assigned to the stockItem attribute and a UOMModeDisplayer is assigned to the unitOfMeasureMode attribute.

Table Editor

Each attribute value is examined by the table during instantiation and default table editors are assigned for the data type belonging to the attribute. If the attribute is a Quantity, then a QuantityTableEditor is assigned; if the attribute is a Boolean, then a BooleanTableEditor is assigned, and so on. If the attribute requires specialized editing, then a table editor should be assigned to the attribute manually. In our example, a StockItemTableEditor is assigned to the stockItem attribute. The StockItemTableEditor is declared at the class level so it can modified by class code.

Editable

Should be assigned true if the column allows editing, false otherwise.

Displayers Displayers are a hierarchy of classes responsible for formatting the information of an object into a displayable string. Each displayer must implement the method getDisplayText(). The package oracle.retail.sim.closed.swing.displayer in the client project contains numerous useful generic displayers.

Creating a New Displayer Use the following procedure to create a new displayer as needed:

1. Check all previously existing displayers. Many of the generic displayers can handle formatting different objects (such as AttributeDisplayer, DualAttributeDisplayer and DefaultDisplayer).
2. Design the new displayer and extend the appropriate superclass (often AbstractDisplayer).
3. Implement the getDisplayText() methods.

Example 7–9 StoreDisplayer

```
public class StoreDisplayer extends AbstractDisplayer {
  private String separator = " - ";
  public String getDisplayText(Object object) {
      if (object == null) {
          return StringConstants.EMPTY;
      }
      if (object instanceof BuddyStore) {
          BuddyStore store = (BuddyStore) object;
          return store.getBuddyId() + separator + store.getBuddyName();
      }
      if (object instanceof Store) {
          Store store = (Store) object;
          return store.getId() + separator + store.getName();
      }
      if (object instanceof SimStore) {
          SimStore store = (SimStore) object;
          return store.getId() + separator + store.getName();
      }
      return object.toString();
  }
}
```

The Displayable Interface There is an interface named Displayable with a single method toDisplayString(). This is intended to be implemented by wrappers, business objects and any other type of object that might need to have a display value, but which the toString() method is primarily reserved for debugging. Currently, Store is the only object to implement Displayable. The UI framework is already set up to recognize and use Displayable. For example, if you drop several Displayable objects into an RComboBoxEditor, it automatically calls toDisplayString() to determine what to display.

Example 7–10 StoreDisplayer

```
public class Store extends BusinessObject implements Displayable {
    public String toDisplayString() {
        return id + " - " + name;
    }
    public String toString() {
        StringBuilder buffer = new StringBuilder();
        buffer.append("Store: [");
        buffer.append("Id: ").append(getId());
        buffer.append(";  Name: ").append(getName());
        buffer.append(";  Language: ").append(getLanguage());
        buffer.append(";  Country: ").append(getCountry());
        buffer.append(";  Currency: ").append(getCurrencyCode());
        buffer.append(";  Sim flag: ").append(getSimFlag());
        if (timezone != null) {
            buffer.append(";  Timezone: ");
            buffer.append(timezone.getDisplayName());
        }
        buffer.append("]");
        return buffer.toString();
    }
}
```

TableEditors Like the editors designed to be used on screens, table editors are advanced widgets designed for use within table cells. The SimTable uses the SimTableEditor interface to control editing within SimTable cells.

- CustomUINCreateDialogTableEditor is included in the example code set as an implementation of a normal table editor.
- CustomUINCountTableEditor is included in the example code set as an implementation of a table editor that opens a dialog box.

There are many generic table editors located within the client project in oracle.retail.sim.closed.swing.tableeditor package. If these do not meet requirements, there are SIM-specific table editors within the client project in the package oracle.retail.sim.close.swingclient.tableeditor. If these do not meet requirements, a custom table editor must be written.

> **Note:** No specific documentation on creating table editors is planned for this document. Table editors are complicated objects that require very precise design. Design and code review for any new table editors should be performed by a senior developer.

Table Wrappers and Value Objects

When creating a table definition, the getDataClass() is used to specify what object is being displayed. Often, the definition of a row does not exactly match a business object. Sometimes a row might require two business objects. Sometimes the API of the business object does not match what is desired for display. Design of the business layer and its objects should be done strictly with the idea of capturing functional requirements, relationships and behavior, and not necessarily with how the business layer and its objects are gathered and displayed on the screen.

There are two methods of dealing with a row that does not cleanly map to a business object:

Wrappers

Wrappers are UI objects that wrap one or more business objects into a single API that the table can easily access and modify. This creates an isolated layer between the table and business object where the UI client code can live that does not belong to the behavior of the business object.

The following is a poor example, as it directly uses the business object ReceiptLineItem:

Example 7–11 DirectDeliveryDetailPanel

```
private static class DirectDeliveryItemDefinition implements SimTableDefinition {
   public Class getDataClass() {
    return ReceiptLineItem.class;
  }
```

The following is a good example, as it uses a wrapper for the direct delivery line item that represents an interface to the table:

Example 7–12 DirectDeliveryDetailPanel

```
private class DirectDeliveryItemDefinition extends SimTableDefinition {
    public Class getDataClass() {
        return DirectDeliveryLineItemWrapper.class;
}
```

A wrapper simply wraps another object or objects and has methods that are designed for the table and its column. The wrapper determines what to do with the code and passes it on to the business objects. Note how the wrapper contains both a stock item and the serial number values to add. This makes it convenient to create one table row API that accesses both objects:

Example 7–13 CustomUINCreateWrapper

```
public class CustomUINCreateWrapper {
    public static final String STOCK_ITEM_PROPERTY = "stockItem";
    public static final String DESCRIPTION_PROPERTY = "description";
    public static final String UIN_COUNT_PROPERTY = "serialNumberCount";
    private StockItem stockItem;
    private List<SerialNumberValue> serialNumbers = CollectionUtil.newArrayList();
    public StockItem getStockItem() {
        return stockItem;
    }
```

Value Objects

Value Objects are end-to-end objects that represents a summary or partial view of one or more objects. Value objects should be designed to clean representations of the data matching exactly where they are going to be used. Value objects do not execute rules or contain setters. They contain only getter methods and doSet() methods that should only be used by the DAO layer. They also tend to declare only basic data they need and not contain complete objects. A value object should only be used within the code when the row of the table is not going to be edited, as the value object is considered unchangeable. The ItemVO is a good example of this pattern. This value object is designed to primarily be used on the ItemLookupPanel (which doesn't edit any information) and so its information matches the screen.

Example 7–14 ItemVO

```
public class ItemVO implements Serializable, ItemDescription {
    private String id = "";
    private String shortDescription = "";
    private String longDescription = "";
    private SupplierVO supplierVO = null;
    private String departmentName = "";
    private String className = "";
    private String subclassName = "";
    private boolean isRanged = true;
    private ItemType itemType = ItemType.ITEM;
    public ItemVO(String id) {
        this.id = id;
    }
    public String getId() {
        return id;
    }
    public String getShortDescription() {
        return shortDescription;
    }
// Remainder of code…}
```

In this example, note how the ItemVO contains departmentName instead of an ID or the full merchandise hierarchy node. This is because the name is the only item required where the VO is used. The same with adding the flag isRanged(), which a normal item does not contain. This makes the item much smaller and easier to transmit across the service call.

Triggering User Interface Events Sometimes, a user might want to perform an action and execute some logic when an event occurs within the components on the screen. Under these circumstances, there is a specific framework in place to accomplish this. Whenever possible, avoid using standard Swing listeners to accomplish these kinds of tasks, instead using REventListeners and RActionEvents:

Regular Editor Actions

Do the following to receive and process actions. Almost all actions of this nature take place within the Panel code.

1. Register an action on an editor.

   Every editor within the system implements RetailEditor that contains the method registerAction(). There are two parameters: the listener and the command. When the contents of the editor change, the listener is notified with the command. Actions are normally registered within the initializePanel() method of the Panel.

Example 7–15 DirectDeliveryCreatePanel

```
private RSearchFieldEditor itemEditor =
                SimEditorFactory.createNonRangingItemSearchFieldEditor("Item");
private RSearchFieldEditor supplierEditor =
SimEditorFactory.createSupplierSearchFieldEditor();
private static final String ITEM_MODIFIED = "Item.modified";
private static final String SUPPLIER_MODIFIED = "Supplier.modified";

private void initializePanel() {
    // Additional Code Here
    itemEditor.registerAction(this, ITEM_MODIFIED);
    supplierEditor.registerAction(this, SUPPLIER_MODIFIED);
    // Additional Code Here}
```

2. Receive and parse the action.

All RActionEvents that are generated are sent to the method performActionEvent(). The method should parse out which action took place in the standard pattern and delegate to a method to execute the logic. This pattern is preferable to creating inner class listeners on the fly because there is one centralized place within the panel to find where all actions are being delivered. This pattern also aids debugging as well.

Example 7–16 DirectDeliveryCreatePanel

```
public void performActionEvent(RActionEvent event) {
    String command = event.getEventCommand();
    try {
        if (command.equals(TYPE_SELECTED)) {
            doRadioSelectionModified();
        } else if (command.equals(ITEM_MODIFIED)) {
            doItemModified();
        } else if (command.equals(ITEM_SUPPLIER_MODIFIED)) {
            doItemSupplierModified();
        } else if (command.equals(SUPPLIER_MODIFIED)) {
            doSupplierModified();
        } else if (command.equals(PO_MODIFIED)) {
            doPurchaseOrderModified();
        } else if (command.equals(PACK_ITEM_MODIFIED)) {
            doPackItemModified();
        } else if (command.equals(PACK_MODIFIED)) {
            doPackModified();
        } else if (command.equals(PACK_SUPPLIER_MODIFIED)) {
            doPackSupplierModified();
        }
    }
 catch (Throwable e) {
        displayException(e);
    }
}
```

3. Implement the action logic.

Implement the private method that executes the logic associated with the editor.

Table Editor Actions

Editors within tables can be listened to in order to receive events, but they are handled in an entirely different manner.

Use the addTableEditorListener() method to add a listener to the table editor. A table editor can either be declared as a class variable so it is handy or can be retrieved from the table itself based on the type of property that is being modified.

Build a listener. Use a build<name>Listener method to get the code separated into a convenient method rather than declaring inline where it might obfuscate what is taking place. In the following example, every time the stock item value changes within the table, the validateEnabledState() method is called within the panel:

Example 7–17 ReturnDetailPanel

```
private StockItemTableEditor stockItemTableEditor = new StockItemTableEditor();
private void initializePanel() {
   stockItemTableEditor.addTableEditorListener(buildStockItemListener());
}
private SimTableEditorListener buildStockItemListener() {
    return new SimTableEditorListener() {
        public void performTableEditorEvent(SimTableEditorEvent event) {
            validateEnabledState();
        }
    };
}
```

Screen Models

Screen models hold onto the state of data or information and interact with the business layer of the system for the panel of UI widgets. The following is an example of building a screen model (using the CustomUINCreateModel.java found in the example code):

1. Declare the model. The model must extend SimScreenModel. Declare any variables, usually the data the model represents. No constructor is necessary for a model as there are never any parameters passed into a model constructor.

   ```
   public class CustomUINCreateModel extends SimScreenModel {
       private String CUSTOM_UIN_SERVICES_KEY = "CUSTOM_UIN_SERVICES_KEY";
   ```

2. Add all the functional methods required by the panel to communicate with the server or manipulate data on a logic level. Here is an example of common methods found in a model.

   ```
   public void saveSerialNumbers(List<CustomUINCreateWrapper> wrappers) throws
   Exception {
    List<CustomSerialNumber> serialNumbers = CollectionUtil.newArrayList();
    for (CustomUINCreateWrapper wrapper : wrappers) {
        for (SerialNumberValue value : wrapper.getSerialNumbers()) {
            CustomSerialNumber serialNumber = new CustomSerialNumber();
            serialNumber.setId(value.getUINDetailId());
            serialNumber.setStatus(value.getStatus());
            serialNumbers.add(serialNumber);
        }
    }
    getCustomUINService().moveSerialNumbersToInStock(serialNumbers);
   }
   ```

Features of the Model The SimScreenModel superclass contains several features that can be used by all subclasses.

Table 7–11 Model Features

Feature	Description
getUser()	Retrieves the current user logged into the application.
getStore()	Retrieves the current store the user is logged into.
getAllowedStores()	Retrieves a list of stores the user is allowed to access.
getTimeZone()	Retrieves the time zone of the current store.
hasPermission()	Returns true if the user has the specified permission.
hasDataPermission()	Returns true if the user has the specified permission for a particular piece of data.
releaseLock()	Release an activity lock on the transaction being used.
confirmLock()	Confirms the current user still holds the activity lock on the transaction being used.
obtainLock()	Obtains an activity lock for a user on a transaction.
buildExceptionWithValue( )	Creates a UIException where there is a substitute value within the message.

Dialog Windows

There are a few different usages of dialog boxes in SIM. The most common usage is to display errors, messages and confirmations during the workflow process. Other usages include popup search dialogs (item & supplier) and entering data.

Error & Message Dialogs The developer never instantiates or creates an error or message dialog. These types of dialogs are handled entirely by the framework. Models cannot display errors because they only represent logic, so all exceptions must be propagated to the Panel or Screen level where helper methods exist to display the exceptions. The displayMessage() and displayWarning() methods take a string and display a message window. These methods are only available at the panel level as the screen level is thin enough to not need these methods. The following example in ItemTicketListPanel uses several different helpers all in one method:

Example 7–18 ItemTicketListPanel

```
private void printTickets() throws Exception {
    try {
        List<ItemTicketWrapper> wrappers =
itemTicketTable.getAllSelectedRowData();
        String message = model.printTickets(wrappers);
        if (message != null) {
            displayMessage(Translator.getMessage(message));
        }
    } catch (NoPrinterDefinedException noPrinterException) {
        displayError(noPrinterException.getMessage());
    } catch (BusinessException exception) {
        displayException(exception);
    } finally {
        itemTicketTable.setRows(model.findItemTickets());
    }
}
```

Confirmation Confirmation dialogs are Ok/Cancel or Yes/No dialogs that allow the user to make decisions. Confirmation dialogs are not instantiated. A utility class (RConfirmUtility) is supplied to make the display of confirmation dialogs easier. The confirm() method returns **true** if the option was confirmed; **false** if it was not. The following examples show some different usages of the confirmation utility:

Example 7–19 ReturnDetailPanel

```
protected boolean cancelReturn() throws Exception {
    if (RConfirmUtility.confirmWithOkCancelType("Delete Return Confirmation",
                SimClientErrorKey.RETURN_MISSING_QUANTITY)) {
        model.cancelReturn();
        return true;
    }
    return false;
}
```

Example 7–20 MacroSequenceListPanel

```
public void handleApplyClassList() throws Exception {
    if (model.isSequenceCoherent()) {
        if (RConfirmUtility.confirm("Confirmation",
                                    SimClientErrorKey.SEQUENCE_GENERATE_LOCATION_
CONFIRM)) {
            LocationArea locationArea = LocationArea.BACKROOM;
            if (RConfirmUtility.confirm("Select Area",
                    SimClientErrorKey.SEQUENCE_SHOPFLOOR_OR_BACKROOM, "Shopfloor",
"Backroom")) {
                locationArea = LocationArea.SHOPFLOOR;
            }
            try {
                model.applyClassList(locationArea);
            } catch (BusinessException exception) {
                displayException(exception);
            }
            locationTable.setRows(model.findLocations());
        }
    }
}
```

Popup Windows All popup windows have RDialog as a super-class. If the customization includes popping up a dialog window to enter data, then by using RDialog as the super-class, the customized code can access several additional useful methods connected to SIM's framework.

Building Wireless Forms

The SIM Wireless handheld client is a Wavelink application. All wireless clients (users in the store) use handheld physical devices to talk to a wireless container (sometimes called the wireless server). Most of the SIM application can be accessed through these handheld devices. In certain areas such as stock counts and product groups, some administrative tasks must be done on the PC as these tasks cannot be performed by the handheld device.

Wireless Application Architecture

The client architecture is broken into five layers, only three of which are worked on during application development:

Wireless Framework This is the wireless framework application consisting of the Wavelink code that starts and executes as a server and handles the actual communication protocol back and forth to the handheld devices.

Form_<name> The framework loads and displays forms on the handheld device. The framework then receives actions from these forms back at the server – similar to a web page. These forms are not coded, but are generated from xml files (also similar to web pages). A form's xml file will be named **screen_<name>** and located in its own directory in the wireless project with the path `bin.generator.screens`.

EventHandler_<name> AbstractEventHandlers are generated along with the forms to receive actions. The EventHandler_<name> class extends the abstract class and actually implements the actions that can be received from the form. The EventHandler classes are coded by the developer to handle the logic for the handheld device.

<functional area>Utility Often, an EventHandler communicates with the rest of the SIM system by using logic available through a utility that covers common logic within a functional area.

EJB Service The EJBs to the regular SIM services are accessed from EventHandlers or Utilities by using the client-side EJB services (either by specifically instantiating or by using ClientServiceFactory).

Forms

Forms contain the page information that is sent back and forth to the actual device.

Event Handler

When a form is created, an AbstractEventHandler_<name>.java file must exist as well. The developer must then code an EventHandler_<name>.java file that matches the abstract handler. This section will outline how an EventHandler relates to a form and some of the general methods that are available to an EventHandler. The following is an example of a form designed in an .xml file for entering an item on an inventory adjustment:

Example 7–21 Screen_InventoryAdjustmentNormalItem

```
<Screen name="InventoryAdjustmentNormalItem">
  <LogicalScreen>
    <field name="itemId" type="string" length="21" />
    <field name="itemDesc" type="string" length="42" />
    <field name="reason" type="string" length="21" />
    <field name="unavailableLabel" type="string" length="10" />
    <field name="unavailableQty" type="string" length="5" />
    <field name="qtyLabel" type="string" length="15" />
    <field name="qty" type="string" length="5">1</field>
    <field name="uom" type="string" length="5" />
    <field name="packSize" type="string" length="5" />
  </LogicalScreen>
<PhysicalScreens deviceclass="dnw">
    <PhysicalScreen seq="0">
      <label y="0" x="0" height="1" width="21"
            style=".heading1">$[wireless.inventoryAdjustment]</label>
```

```
      <label y="2"  x="0"  height="1" width="21" name="itemId"
field="itemId"></label>
      <label y="3"  x="0"  height="2" width="21" name="itemDesc"
field="itemDesc"></label>
      <label y="5"  x="0"  height="1" width="21" name="reason"
field="reason"></label>

      <label y="6"  x="0"  height="1" width="10" name="unavailableLabel"
             field="unavailableLabel"></label>
      <label y="6"  x="10"  height="1" width="6" name="unavailableQty"
             field="unavailableQty"></label>
      <label y="6"  x="16"  height="1" width="5" name="unavailableUnits" />

      <label y="8"  x="0" height="1" width="9" name="qtyLabel"
field="qtyLabel"></label>
      <input y="8"  x="10" height="1" width="6"  name="qty" field="qty" seq="0"
             acceptScan="false" validateKeyEventOnScan="false"/>
      <label y="8"  x="16" height="1" width="5"  name="uom" field="uom" />

      <label y="9"  x="0"  height="1" width="10" name="packSizeLabel" />
      <input y="9"  x="10" height="1" width="6" name="packSize" field="packSize"
             seq="1" acceptScan="false" validateKeyEventOnScan="false"/>

      <scan/>

      <cmdkey y="0" x="0" width="0" height="0" key="&toggle;" name="toggle"
              action="callMethod" target="doToggle"/>
      <cmdkey y="0" x="0" width="0" height="0" key="&exit;" name="Exit"
             action="callMethod" target="doExit" />
  </PhysicalScreen>
</PhysicalScreens>
</Screen>
```

The API of the AbstractEventHandler that was created to match the form is shown in the following example:

Example 7–22 AbstractEventHandler_InventoryAdjustmentNormalItem

```
abstract public class AbstractEventHandler_InventoryAdjustmentNormalItem extends
SimEventHandler {
    abstract protected void onFormOpen();
    abstract protected void onFormClose();
    abstract public void onScan(String data);
    abstract public boolean qty_OnChange(String newValue);
    abstract public boolean qty_OnExit(String newValue);
    abstract public boolean packSize_OnChange(String newValue);
    abstract public boolean packSize_OnExit(String newValue);
    abstract public void doToggle();
    abstract public void doExit();
}
```

OnFormOpen() and OnFormClose() must exist for each AbstractEventHandler regardless of the form:

- OnFormOpen() is executed before the form is displayed to the handheld device. The code that populates the form with data should be placed in this method.
- OnFormClose() is executed when the form is removed from the handheld device.

The onScan() method matches the <scan/> tag in the xml, which indicated a row on the form into which to scan a value. When a value is scanned by the device, the information is passed to the onScan() method as a data parameter. The developer can implement this method in the EventHandler to process the scanned data (in this case, the barcode of an item).

Both qty_OnChange() and qty_OnExit() were created by the <input> tag with the field=qty set within that tag. Since qty was entered as the field, these two methods exist to handle processing when a quantity is entered.

Both packSize_OnChange() and packSize_OnExit() were created by the <input> tag with the field =packSize set within that tag. Since packSize was entered as the field, these two methods exist to handle processing when a pack size is entered.

The doToggle() method was created by the <cmdkey> tag based on the target=value section of the tag. Since doToggle was entered as the target value, this method was created. A cmdkey displays as an option on the screen. The method is executed in the EventHandler when the option is chosen on the screen. This is the same with the doExit() based on its <cmdkey> tag.

SimEventHandler

Each AbstractEventHandler extends from SimEventHandler, a superclass that contains methods for common tasks. These methods should always be used when these tasks must be performed. There are methods for the following:

- Assigning data to forms
- Reading data from forms
- Displaying alerts
- Displaying exceptions
- Checking locks
- Releasing locks
- Showing specialized screens (barcode, Yes/No choice, text input)
- Navigating to other forms

YesNoEventHandler

YesNoEventHandlers are simple choice windows that display a choice and allow the user to pick **Yes** or **No**.

Example 7–23 RequestCancelYesNoHandler

```
public class RequestCancelYesNoHandler extends SimYesNoHandler {
  public void performYes(IApplicationForm currentForm) throws Exception {
    try {
      ItemRequest itemRequest =
ItemRequestWirelessUtility.getContext().getItemRequest();
      if (itemRequest.getLineItems().size() > 0) {
          ItemRequestWirelessUtility.doSave(itemRequest, false);
      }
      currentForm.gotoForm(ItemRequestWirelessKeys.SCREEN_MENU);
      } catch (BusinessException be) {
          currentForm.gotoForm(ItemRequestWirelessKeys.SCREEN_MENU);
      } catch (Exception e) {
          handleException(e, currentForm);
      }
```

```
  }
  public void performNo(IApplicationForm currentForm) throws Exception {
    try {
        currentForm.gotoForm(ItemRequestWirelessKeys.SCREEN_SUMMARY);
    } catch (Exception e) {
        handleException(e, currentForm);
    }
  }
  public String getTitle() throws Exception {
    return getText(ItemRequestWirelessUtility.getTitleKey());
  }
  public String getMessage() throws Exception {
    String id = ItemRequestWirelessUtility.getContext().getItemRequest().getId();
    return getMessage(ItemRequestWirelessKeys.MESSAGE_EXIT_CONFIRM, id);
  }
}
```

The SimYesNoHandler superclass that all YesNoHandlers should extend contains most of the same helper methods as the SimEventHandler class. This means that such functionality as translating text and handling exceptions should always use these helper methods.

The SimYesNoHandler also has the implemented code that returns the Yes and No choice labels for the screen, so the user only has to implement the following four methods for each new YesNoHandler:

- performYes() is executed when the user chooses the **Yes** option.
- performNo() is executed when the user chooses the **No** option.
- getTitle() returns the title to display at the top of the form.
- getMessage() return the query text to display on the form.

Wireless Context

A context represents a repository of data entered or being altered for a particular functional area within the user session. This context is carried in the repository to make it readily accessible between different forms. The InventoryAdjustmentContext is used as an example to trace some of the usages of context. Note that the context object itself simply has a set of data variables.

Example 7–24 InventoryAdjustmentContext

```
public class InventoryAdjustmentContext {
    private InventoryAdjustment inventoryAdjustment;
    private InventoryAdjustmentReason lastAdjustmentReason = null;
    private String lastScannedUIN = null;
    private Quantity scanEnteredQty = null;
    private Quantity computedQty = null;
    private boolean takeFromUnavailableBucket = false;
    private int currentUINIndex = 0;
    public InventoryAdjustment getInventoryAdjustment() {
        return inventoryAdjustment;
    }
   public void setInventoryAdjustment(InventoryAdjustment inventoryAdjustment)
{
        this.inventoryAdjustment = inventoryAdjustment;
    }
    public boolean isTakeFromUnavailableBucket() {
        return takeFromUnavailableBucket;
```

```
    }
    public void setTakeFromUnavailableBucket(boolean takeFromUnavailableBucket)
{
        this.takeFromUnavailableBucket = takeFromUnavailableBucket;
    }
    // Other Getters & Setters
}
```

Access to the context is through the utility for the functional area, which contains a set of static helper methods to create, retrieve and remove the context. Note that the actual context itself is stored within the SimWirelessRepository.

Example 7–25 InventoryAdjustmentWirelessUtility

```
public static InventoryAdjustmentContext getContext() {
    return (InventoryAdjustmentContext)
       SimWirelessRepository.getValue(InvAdjustmentWirelessKeys.CONTEXT);
}

public static InventoryAdjustmentContext createContext() {
    InventoryAdjustmentContext context = new InventoryAdjustmentContext();
    SimWirelessRepository.setValue(InvAdjustmentWirelessKeys.CONTEXT, context);
    return context;
}
public static void removeContext() {
   SimWirelessRepository.removeValue(InvAdjustmentWirelessKeys.CONTEXT);
}
```

Wireless Utilities

Wireless Utilities are static classes that contain numerous helper methods to execute business logic, such as in the examples for context in which the context is created or retrieved, or in the following example where a new inventory adjustment was created. There is only one <name>WirelessUtility for each functional workflow on the handheld device.

The Wireless Utility contains public static helper methods to perform business logic for EventHandlers. Wireless Utilities do not contain any data variables at the class level as the utility does not store state of its data. That responsibility is handled by the context. The following example code shows a utility method for persisting the inventory adjustment:

Example 7–26 InventoryAdjustmentWirelessUtility

```
public static void persistInventoryAdjustment(IApplicationForm currentForm) {
    InventoryAdjustment invAdjustment = getContext().getInventoryAdjustment();
    try {
        if (invAdjustment.isCoherent()) {
            ClientServiceFactory.getInventoryAdjustmentServices()
                .processInventoryAdjustment(invAdjustment);
        alert(currentForm, getTitleKey(), InvAdjustmentWirelessKeys.MESSAGE_
ADJUSTMENT_COMPLETE,
               ItemWirelessKeys.SCREEN_SCAN_BARCODE, false);
        }
```

```
     } catch (BusinessException businessException) {
        String message = getMessage(businessException.getMessage(),
                                    businessException.getParameters());
        alert(currentForm, getTitleKey(), message, getItemScreen(), true);
     } catch (Exception e) {
        handleException(e, currentForm);
     }
}
```

Form Paging

The ScreenPageManager and PageableEventInterface are used to create selection screens that go on for more than one visual screen worth of information on the handheld device. A **next** and **previous** command option exists on the base of the form to scroll through the pages.

The ScreenPageManager is a wireless framework class that handles a lot of the formatting and displaying of the options on a form and controls going forward and backward through the pages. Developers should never modify this class.

Primarily, developers access this information by having an EventHandler for a form implement the PageableEventInterface. The ScreenPageManager uses this defined API to control the paging.

Example 7–27 PageableEventInterface

```
public abstract Map getMenuItemMap();
public abstract String getScreenName();
public abstract IApplicationForm getCurrentForm();
public abstract String[] getPageFieldNames();
public abstract String[] getSpecialFieldValues();
```

The ItemRequestSelectRequest form displays a list of requests for the user to select from. These requests might not all fit on one form, so this EventHandler implemented the PageableEventInterface. In the implementation of getMenuItemMap(), a Map of ItemRequests is returned from the Context to be displayed as the list of options to select. The getScreenName() method returns the title of the screen.

Example 7–28 EventHandler_ItemRequestSelectRequest

```
public Map getMenuItemMap() {
    return (Map) SimWirelessRepository.getValue(ItemRequestWirelessKeys.ORDER_
MAP);
}
public String getScreenName() {
    return ItemRequestWirelessKeys.SCREEN_SELECT_REQUEST;
}
```

The method getCurrentForm() is implemented by SimEventHandler, so all EventHandlers automatically implement that method. The getPageFieldNames() method returns a list of names to assign to the options displayed on the screen. So in the following example, nine options (or item requests) are displayed with **order0** through **order9** assigned as their name:

Example 7–29 EventHandler_ItemRequestSelectRequest

```
private static final String[] orderFieldNames = { "order0", "order1", "order2",
"order3", "order4", "order5", "order6", "order7", "order8", "order9" };

 public String[] getPageFieldNames() {
```

```
    return orderFieldNames;
}

 public String[] getSpecialFieldValues() {
    return null;
}
```

When the option is selected, the Wavelink framework calls back to a method named after the options names assigned in getPageFieldNames(). In this case, selectOrder0(), selectOrder1(), and so on. This is where the developer can place code to handle the selection. There are no examples of getSpecialFieldValues(). This method returns null in all EventHandlers that implement the interface.

Example 7–30 EventHandler_ItemRequestSelectRequest

```
public void selectOrder0() {
    selectOrder(0);
}
public void selectOrder1() {
    selectOrder(1);
}
// Rest of orderFieldNames…public void selectOrder9() {
    selectOrder(9);
}
```

Once the methods are defined, starting the page display is easy. In the onFormOpen() method, if there is a list available to display selections for, then retrieve the page manager and call initializeScreen(). This sets up and displays the first page of selection options. It should be the last method executed in onFormOpen():

Example 7–31 EventHandler_ItemRequestSelectRequest

```
protected void onFormOpen() {
  try {
      LogService.debug(this, getClass().getSimpleName() + ".onFormOpen()");
      setFormText(FIELD_INSTRUCTIONS, getLabel("Select Item Request"));
      if (getMenuItemMap() == null || getPageManager() == null) {
          initializeOnEntry();
      } else {
          getPageManager().initializeScreen();
      }
  } catch (Exception e) {
      handleException(e);
  }
}
```

Customizing Wireless Forms

This section contains some tips on customizing wireless code. Customizing wireless code requires modifying source code in SIM. This needs to be handled with care.

Coding Guidelines

- The wireless clients should always access services through the ClientServiceFactory class.
- Do not alter existing context classes. Create new ones instead and store in SimWirelessRepository.

- When at all possible, design the modifications to be new forms or replacements to current forms instead of modifying current forms.
- Similar to forms/screens, attempt to write new utility classes rather than modifying current ones.

By following the previous guidelines, the only code that should need to be modified in SIM is the code that directs one form to a newly designed form – keeping the amount of rework at a minimum for future releases.

Creating a New Context

If you need to store additional state data during a session, create an entirely new context to store the information. The newly created contact context must then be placed in the SimWirelessRepository. Next, create a custom wireless utility to handle interaction with the context, including placing the context in the SimWirelessRepository.

```
public class MyCustomContext() {
    { . . . . } // Context code.
 }
```

Creating a New Utility

If new functionality is desired in the wireless application that needs to access a new context or access new services, then the developer should create an entirely new utility to use from the eventhandlers to execute this code.

If you need ready access to a utility method or need to override the functionality of a utility method, then subclass the utility in question with your new utility:

Example 7–32 MyCustomUtility

```
public class MyCustomStockCountUtility extends StockCountWirelessUtility {
    public static MyCustomContext getContext() {
      return (MyCustomContext)
SimWirelessRepository.getValue(MyCustomContextKeys.CONTEXT);
    }
    public static MyCustomContext createContext() {
        MyCustomContext context = new MyCustomContext ();
        SimWirelessRepository.setValue(MyCustomContextKeys.CONTEXT, context);
        return context;
    }
    public static void removeContext() {
       SimWirelessRepository.removeValue(MyCustomContextKeys.CONTEXT);
    }
    [ . . . . ] // New Utility Code
}
```

Altering Code in an EventHandler

Because of the manner that the Wavelink code functions, eventhandlers cannot be easily replaced or sub-classed. Instead, the eventhandler must be removed from the JAR, re-coded, and placed in a custom JAR earlier in the classpath. These modifications are not guaranteed to function correctly with newer releases.

Altering Code in a Form

Altering a Wavelink form is a complicated process:

1. Alter the form_<name> source code.
2. Alter the form_dnw_<name> source code.
3. Alter the AbstractEventHandler_<name> source code.
4. Alter the EventHandler_<name> source code.

All of these files must be placed back in the custom JAR that is first in the classpath.

Significant Classes to Understand

Before attempting to modify or create new handheld Wavelink forms, read all documentation supplied by Wavelink. The following is a list of critical framework classes:

- CommandListHelper
- CommandListInterface
- InputTextInterface
- LocaleWirelessUtility
- PageableEventInterface
- PrintSelectInterface
- ScreenPageManager
- SimBasicHandler
- SimEventHandler
- SimWirelessRepository
- SimYesNoHandler
- WirelessExceptionManager
- WirelessPrintService
- WirelessUtility
- WirelessValidation
- YesNoEventInterface

Exceptions and Logging

This section covers exceptions and logging in SIM so that if code is customized, the exception handling framework can be utilized.

Exceptions

SimServerException

The SimServerException class represents an exception originating on the server. It contains an ID counter and a timestamp field. This ID counter starts at one (**1**) and increases until the server is restarted or the maximum integer value is reached and then it resets to one (**1**) again. The ID is used to uniquely identify an exception within a server log. This class can be instantiated around another exception, such as an SQLException.

DowntimeException
A DowntimeException occurs on the client when communication to the server fails or a severe problem on the server takes place.

BusinessException
This type of execution is thrown from either the client or server whenever the code encounters an attempt to perform some action that is defined as invalid by the functional business requirements. Rules and business objects primarily throw BusinessExceptions. BusinessExceptions are not considered severe errors and allow the system to continue to operate after the exception takes place.

UIException
A UIException is used strictly on the client and indicates a failure in the GUI framework or a general failure in UI processing.

Exception Handling

The DAO Layer
All DAO interfaces should throw SimServerException. These exceptions should not be logged in the DAO layer. This is handled by the EJB before propagating the exception to the client. The database framework generates most SimServerExceptions, though sometimes logic requires manually throwing a SimServerException. The following is an example of a DAO interface declaration and a SimServerException thrown manually:

Example 7–33 ProductGroupDao

```
void insert(ProductGroup productGroup) throws SimServerException;
```

Example 7–34 ProductGroupOracleDao

```
public void insert(ProductGroup productGroup) throws SimServerException {
  if (productGroup.getId() != null) {
     throw new SimServerException("Unable to insert product group with an existing
identifier!");
  }
  productGroup.doSetId(getNextGroupId());
  execute(new ParametricStatement(RkProductGroupDataBean.INSERT_SQL,
            fromObjectToBean(productGroup).toList(true)));
  if (productGroup.isAllItems()) {
      return;
  }
  insertProductGroupDetails(productGroup);
}
```

The Service Layer

The service layer framework is written so that Exception, SimServerException, BusinessException and DowntimeException are all thrown by the service and the EJB, such that the exception is not wrapped when it arrives at the client. Services should all be declared to throw a simple Exception. No logging needs to be handled manually in the code. The framework code handles logging the exceptions before throwing them to the client. The following is an example of a service declaration and a BusinessException thrown manually from within the service layer. DowntimeExceptions are generally only thrown when an unexpected Throwable is caught from the code (such as a NullPointerException) or the EJB stub is no longer communicating.

Example 7–35 ReplenishmentServerServices

```
public PickList updatePickList(PickList pickList) throws Exception {
    PickListUpdateCommand command = CommandFactory.createPickListUpdateCommand();
    command.setPickList(pickList);
    command.execute();
    return command.getPickList();
}
```

Example 7–36 PickListUpdateCommand

```
protected void doExecute() throws Exception {
  PickList dbPickList =
DAOFactory.getPickListDao().selectPickList(pickList.getId());
  if (dbPickList == null) {
      throw new BusinessException(ErrorKey.PICK_LIST_LIST_MUST_EXIST);
  }
  if (dbPickList.getStatus() == PickListStatus.CANCELED) {
      throw new BusinessException(ErrorKey.PICK_LIST_CANCELLED_ERROR);
  }
  if (dbPickList.getStatus() == PickListStatus.CLOSED) {
      throw new BusinessException(ErrorKey.PICK_LIST_CLOSED_ERROR);
  }
  // Rest Of Method
}
```

The UI Layer

On the client side, it is preferable that a BusinessException be thrown whenever business logic reaches an error state. All exceptions are caught and handled by the framework.

Throwing an Exception

When throwing an exception within the PC application, throw a BusinessException with the appropriate message. The exception should always be propagated to the last place that handled an event in order to cleanly break the execution flow.

Example 7–37 ItemTicketListModel

```
public void updateStockOnHand(ItemTicket itemTicket) throws Exception {
  Quantity stockOnHand = itemTicket.getItem().getAvailableStockOnHand();
  if (stockOnHand.intValue() < 1) {
      throw new BusinessException(SimClientErrorKey.ITEM_NOT_UPDATED);
  }
  // Remainder of Method
}
```

Catching an Exception

Exceptions should always be propagated and caught at the screen layer if triggered by a menu button, or in the panel layer if triggered by an editor event. The helper method displayException() should always be used to handle the error correctly, whether in a screen or panel. The following example allows each of the panel handle() methods to simply throw an exception and allows the screen to handle it.

Example 7–38 ItemTicketListScreen

```
public void navigationEvent(NavigationEvent event) {
    String command = event.getCommand();
    try {
        if (command.equals(SimNavigation.PRINT_TICKETS)) {
            panel.handlePrintTickets();
        } else if (command.equals(SimNavigation.CREATE)) {
            panel.handleCreate();
        } else if (command.equals(SimNavigation.UPDATE_SOH)) {
            panel.handleUpdateStockOnHand();
        } else if (command.equals(SimNavigation.DELETE)) {
            panel.handleDelete();
        }
    } catch (Throwable exception) {
        displayException(panel, event, exception);
    }
}
```

An alternate case to handling exceptions at the screen layer occurs when the code must continue to execute normally after the exception is displayed, usually to clean up or reset some information. In the following example, the table must be refreshed whether or not an error occurred:

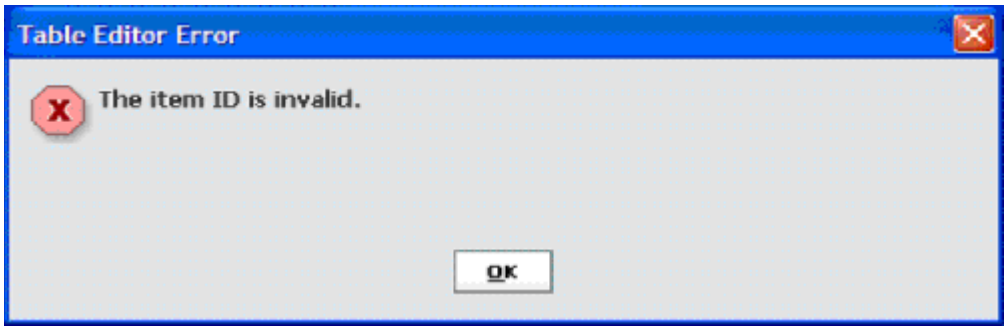
Example 7–39 ItemTicketListPanel

```
public void handleUpdateStockOnHand() throws Exception {
    if (itemTicketTable.getSelectedRowCount() == 0) {
        displayError(SimClientErrorKey.NO_ROWS_SELECTED);
        return;
    }
    // Code removed from example
    List<ItemTicketWrapper> wrappers = itemTicketTable.getAllSelectedRowData();
    for (ItemTicketWrapper wrapper : wrappers) {
        try {
            model.updateStockOnHand(wrapper.getItemTicket());
        } catch (BusinessException exception) {
            displayException(exception);
        }
    }
    itemTicketTable.refreshTable();
}
```

Processing an Exception

BusinessExceptions and UIExceptions are caught and their message displayed in a popup window that locks the application. When the window is exited, control should be returned to the screen/area in which the error occurred. If the UIException has an ErrorSeverity of FATAL, then a Fatal Window is displayed and the user is returned to the main login screen of the application.

Figure 7–2 Table Editor Error Screen



SimServerExceptions are thrown from the server layer and regardless of the details of the exception, only one message displays.

It includes an error number and the date and time that the error occurred. This can be used to find the error in the server exception log. Click **Details** to display the contents of the main exception.

Figure 7–3 Severe Error Screen

Logging

Logging allows the developer to write information about errors or processes to a file. Errors on the server side are automatically logged by the EJB class regardless of whether or not they come from the DAO layer or business layer. Therefore, errors that are logged in local code end up being logged twice if the user logs them.

LogService

If it is necessary to log an error in the business layer or DAO layer, this is done through the LogService class, which provides numerous static methods that hide the implementation away from the end developer. The SimStore business object in the following example logs a message if it fails to load the buddy stores.

Note: Certain API such as Java Open Transaction Manager (JOTM) use Apache Log4J as their default logging. Therefore, Log4J will also need to be configured.

Example 7–40 SimStore

```
private void doLoadStoreLists() {
    if (storesNotLoaded) {
        try {
            StoreServices storeServices = ClientServiceFactory.getStoreServices();
            buddyStores
                =
CollectionUtil.newSortedSet(storeServices.findBuddyStores(store.getId()));
            autoReceiveStores
                =
CollectionUtil.newSortedSet(storeServices.findAutoReceiveStores(store.getId()));
            storesNotLoaded = false;
        } catch (Exception exception) {
            LogService.error(this,"SimStore could not load store lists",
exception);
        }
    }
}
```

Debugging

LogService provides the ability to log messages at the info, debug and warn level as well. All debugging messages should be logged through LogService as well. In the following example, the e-mail dispatcher logs a warning if no permission is associated with the e-mail.

Example 7–41 EmailDispatcher

```
public static void sendEmailAlert(EmailAlert emailAlert) {
  String fromAddress = RkConfigManager.getString(RkConfigManager.EMAIL_FROM_NAME);
  if (StringHelper.isNullOrEmpty(fromAddress)) {
      LogService.error("EmailDispatcher", "Invalid email from-address");
      return;
  }
  String permissionName = emailAlert.getPermissionName();
  if (StringHelper.isNullOrEmpty(permissionName)) {
      LogService.warn("EmailDispatcher", "Email alert is not associated to a
permission.");
      return;
  }
  // Remainder Of The Code
}
```

The business rule executed when the logic attempts to create a new line item for a return will log a debug level message if the item is not supplied by the correct source.

Example 7–42 ReturnCreateLineItemRule

```
public static void execute(Return stockReturn, StockItem stockItem) throws
BusinessException {
  Source source = stockReturn.getDestination();
  if (source == null) {
      throw new BusinessException(ErrorKey.RETURN_NO_DESTINATION);
  }
  if (stockReturn.isVendorReturn()) {
      try {
        ItemServices services = ClientServiceFactory.getItemServices();
        if (services.isItemSuppliedBySupplier(stockItem.getId(), source.getId()))
{
          return;
```

```
      } catch (Throwable exception) {
          LogService.debug(ReturnCreateLineItemRule.class, "Failed rule.",
exception);
      }
      throw new BusinessException(ErrorKey.RETURN_NO_SOURCE);
  }
  // Remainder of rule code
}
```

Logs

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 is renamed (for example, sim.log will be renamed to **sim.log.1**) and new log messages are written to a new file (for example, sim.log). If there are already rolled-over logs, they are 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.

Client Side Logs

On the client, logs are sent to the console and to the file <location of sim client>\bin\log\sim.log.

Logging is configured in client/resources/log4j.xml. This file defines which kinds of messages are logged and where they are logged.

Server Side Logs

On the server, logs are saved to <location of server>\bin\log\sim.log. Log messages are also displayed in the console.

Exception Format

The following example demonstrates a formatted and logged service exception. The first line contains the EJB and method name of the exception failure. The second line contains ERROR with an ID (in this case **1**). This ID is displayed on the client side for reference. It is followed by the user ID of the transaction user, a timestamp, the type of exception, the primary message, and the primary message of the root cause. Following that is the stack trace of the exception.

```
ERROR 02:14:05.728 [ejb.ItemEjb] findItemVOs: Exception occurred during service
invocation <
ERROR-1 User: 15000 Time: 11/27/06 2:14 PM Type: ApplicationException Message:
This is an example exception. Root Cause: Application is in illegal state.>
oracle.retail.sim.closed.common.ApplicationException: This is an example
exception. at
oracle.retail.sim.closed.item.ItemServerServices.findItemVOs(ItemServerServices.ja
va:37) at
oracle.retail.sim.closed.item.ItemHelper.findItemVOs(ItemHelper.java:55) at
oracle.retail.sim.closed.item.ejb.ItemEjb.findItemVOs(ItemEjb.java:127) at
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39) at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
25) at
java.lang.reflect.Method.invoke(Method.java:585) at
com.evermind.server.ejb.interceptor.joinpoint.EJBJoinPointImpl.invoke(EJBJoinPoint
Impl.java:35) at
```

```
com.evermind.server.ejb.interceptor.InvocationContextImpl.proceed(InvocationContex
tImpl.java:69) at
com.evermind.server.ejb.interceptor.system.DMSInterceptor.invoke(DMSInterceptor.ja
va:52) at
com.evermind.server.ejb.interceptor.InvocationContextImpl.proceed(InvocationContex
tImpl.java:69) at
com.evermind.server.ejb.interceptor.system.TxSupportsInterceptor.invoke(TxSupports
Interceptor.java:37) at
com.evermind.server.ejb.interceptor.InvocationContextImpl.proceed(InvocationContex
tImpl.java:69) at
com.evermind.server.ejb.interceptor.system.DMSInterceptor.invoke(DMSInterceptor.ja
va:52) at
com.evermind.server.ejb.interceptor.InvocationContextImpl.proceed(InvocationContex
tImpl.java:69) at com.evermind.server.ejb.StatelessSessionEJBObject.OC4J_
invokeMethod(StatelessSessi onEJBObject.java:86) at ... 24 more
```

Modifying Data Transport to External Systems

We can continue the previous example of Generic Retailers Inc, adding the new attribute Season to the items within the system. Besides the SIM side as given in earlier examples, this information could also arrive from an external system and might need to be sent to external systems. So in this case, the following steps are taken to support the new data:

1. Update DEOs
2. Update Injectors
3. Update Consumers
4. Update Stagers
5. Update Publishers

This customization document does not cover any modifications to code or processes external to SIM, or in other words, how to update the external system. As such, this document does not cover updating RIB configurations and payloads should the external system be linked using the RIB.

Process of Receiving an External Message

Do the following to receive an external message:

1. The external message is received by an Injector.
2. The Injector class converts the message to a Data Exchange Object.
3. The Data Exchange Object (DEO) is written into the STAGED_MESSAGE table.
4. The Data Exchange Object (DEO) is read from the STAGED_MESSAGE table by a polling timer and given to a Consumer object to be processed.
5. The Consumer object uses the DEO to do processing and update SIM information.

Process of Sending an External Message

Do the following to send an external message:

1. Business processes use a Stager class to convert business data into a DEO.
2. The DEO is written into the STAGED_MESSAGE table.

3. The DEO is read from the STAGED_MESSAGE table by a polling timer and given to a Publisher object to be processed.
4. The Publisher object converts the DEO to external information and sends the information to an external system.

Update DEOs

The first step of customizing the external message process is to update the DEOs (or Data Exchange Objects) to contain the added attributes. ItemHdrDEO (incoming message containing the basic item information) and ReturnToVendorLineItemDEO (outgoing message containing item information on a return) will be used as examples of this process.

Since this is new information on already existing messages, the original DEOs should be sub-classed and the new attributes added much the same way as was done for business objects and value objects:

```
public class GriItemHdrDEO extends ItemHdrDEO {
    private String season;
    public GriItemHdrDEO() {
        super();
    }
    public String getSeason() {
        return season;
    }
    public void setSeason(String season) {
         this.season = season;
    }
}
```

DEOs are instantiated through a DEOFactory. This factory uses a configured implementation to create the objects. The next step is to subclass the default implementation (DEOFactoryImpl) to override the methods that create the modified DEOs:

```
public class GriDEOFactoryImpl extends DEOFactoryImpl {
  public ItemHdrDEO createItemHdrDEO() {
      return new GriItemHdrDEO();
  }
  public ReturnToVendorLineItemDEO createReturnToVendorLineItemDEO() {
      return new GriReturnToVendorLineItemDEO();
  }
}
```

To configure the factory, simply modify common.cfg with the classpath of your new code:

```
DEO_FACTORY_IMPL=gri.custom.code.GriDEOFactoryImpl
```

Update Injectors

To update the payload-to-DEO injection mapping in SIM, a developer must extract the actual source code for SIMMessageMapperUtil, then modify the original source code, re-jar the correct project and re-sign everything for deployment. This process should be left up to a system administrator.

The SIMMessageMapperUtil is the class responsible for mapping payloads to DEOs. Find the appropriate map() method and alter the contents of the method:

```
private static ItemHdrDEO convert(ItemHdrDesc itemHdrDesc) {
   ItemHdrDEO itemHdrDEO = DEOFactory.createItemHdrDEO();
   itemHdrDEO.setItemId(itemHdrDesc.getItem());
   //…… remaining original code
   itemHdrDEO.setDefaultWastePercent(itemHdrDesc.getDefaultWastePct());
   itemHdrDEO.setSeason(itemHrdDesc.getSeason());
   // New Attribute
   return itemHdrDEO;
}
```

Update Consumers

The first step of customizing a consumer is to subclass the consumer and add custom logic. In order to deal with the new season attribute, ItemCreateConsumer is customized as an example. Every consumer must implement consume(), and so this is the method that must be overridden with custom code. Calling super.consume() will guarantee that the normal process of consuming external data will be executed and that future releases will continue to work with no code changes.

```
public class GriItemCreateConsumer extends ItemCreateConsumer {
  public void consume(List<DataExchangeObject> deos) throws Exception {
      super.consume(deos);

      GriItemDao griItemDao = new GriItemDao();
      for (DataExchangeObject deo : deos) {
          ItemHdrDEO itemHdrDEO = ((ItemDEO) deo).getItemHdrDEO();
          griItemDao.updateSeason(itemHdrDEO.getItemId(), itemHdrDEO.getSeason());
       }
 }
```

The second step is very similar to other customization patterns: the factory implementation is altered to return the custom consumer and then sim.cfg is updated to point at the new implementation. The custom consumer factory implementation should extend the regular factory implementation and then override the getConsumer() method. After first returning any customized consumers, it can then call the superclass to retrieve a SIM supplied consumer:

```
public class GriMessageComsumerFactoryImpl extends SimMessageComsumerFactoryImpl {
   public SimMessageConsumer getConsumer(SimMessageType messageType) throws
Exception {
       if (messageType == SimMessageType.ITEM_CRE) {
           return new GriItemCreateConsumer();
       }
       return super.getConsumer(messageType);
    }
}
```

To configure the factory, modify sim.cfg with the classpath of your new code:

```
MESSAGE_CONSUMER_FACTORY_IMPL=gri.customer.code.GriMessageConsumerFactoryImpl
```

Update Stagers

Stagers and Publishers are the opposite of Consumers and Injectors. They take data into SIM and prepare it to be sent externally. Using our example of ReturnToVendorLineItemDEO, customization begins with modifying the Stager that deals with staging vendor returns. To do this, a developer must extract the actual source code for VendorReturnStager, then modify the original source code, re-jar the correct project and re-sign everything for deployment. This process should be performed by a system administrator/developer.

Update Publishers

To update the DEO-to-Payload publisher mapping in SIM, a developer must extract the actual source code for SIMMessageMapperUtil, then modify the original source code, re-jar the correct project and re-sign everything for deployment. This process should be performed by a system administrator/developer.

The SIMMessageMapperUtil is the class responsible for mapping DEOs to payloads or other external messages. Find the appropriate map() method and alter the contents of the method.

```
private static void setLineItems(RTVDesc payload, List<ReturnToVendorLineItemDEO>
lineItems) {
   for (ReturnToVendorLineItemDEO lineItem : lineItems) {
      RTVDtl payloadDtl = new RTVDtl();
      payloadDtl.setItemId(lineItem.getItemId());
      payloadDtl.setUnitQty(lineItem.getUnitQty());
      payloadDtl.setFromDisposition(SimArtsConstants.DISPOSITION_TYPE_AVAILABLE);
      payloadDtl.setToDisposition("");
      payloadDtl.setReason(lineItem.getReturnReason());
      payloadDtl.setCustomAttribute(lineItem.getCustomAttribute());
      payload.getRTVDtl().add(payloadDtl);
   }
}
```

Modifying Source Code

Customizing the data transport to external systems currently still requires modifying SIM's source code. To build and deploy source modifications correctly, make a separate project for custom-modified code. Make sure any existing source code files that are going to be altered have the exact same classpath as in the base code. Build a custom JAR with the modified code. This custom JAR should be placed first in the execution classpath so that classes found within the JAR are chosen by the JVM rather than the base classes. This will require un-signing and re-signing all the JARs in the SIM-client application, since all JARs must be signed with the same signature for Web Start to work correctly. Consult the jarsigner documentation from Sun for further information on the JAR un-signing/signing process.

The SIM application is based on a style of code that includes a clear separation of closed and shared concepts. The code that resides in packages that contain **closed** in their classpath is considered essential to the integrity of the product and is not released.

> **Note:** This closed source code should never be modified under any circumstances. Any changes to closed source code will likely make the system unusable and not upgradeable.

Source code that resides in the classpath that contains **shared** in its structure can be modified to accommodate altered functionality. All care should be given to read the appropriate documentation before modifying the code.

Note: Customized source code must be manually integrated during future releases of the product.

Customizing Look and Feel

Customization of the look and feel of the SIM PC application is available directly through the PC GUI and the appropriate privileges. Fonts, colors and icons can be changed to suit the company's desires. Completely new and fresh themes can be created directly through SIM. Refer to the *Oracle Retail Store Inventory Management User Guide* for description of how to use these features.

Visual appearances outside of this spectrum, such as widget shapes or animation, are not open to customization. Note that if completely new workflows are built on the PC, there are no restrictions as to what UI elements are put on the new screens.

The handheld code is not open to look and feel changes. These devices use simple two-tone text-based screens. All customizations are subject to the frameworks provided by Wavelink. See Wavelink documentation for further information.

Customizing Languages

The administrative section of the PC GUI has the ability to customize the translations of any language provided in the release of SIM. Users can create new keys not previously existing in SIM. Understanding of English will be necessary to use this feature of SIM. See the *Oracle Retail Store Inventory Management User Guide* for a description of how to use this feature.

Do the following to add a new language to SIM:

1. Insert a record into RK_TRANSLATION_LOCALE for the new language.

   If Scottish Gaelic is added as a new language, insert the following row into the database. Language Column is the two character ISO 639 code for the language. The country is the ISO 3166 two-character country code. In this case, GB = Great Britain, but it could be left null. Country and variant are not required.

Table 7–12 Insert a Record into RK_TRANSLATION_LOCALE

Column	Value
LOCALE_ID	A new unique ID should be inserted here.
LANGUAGE	GD
COUNTRY	GB
VARIANT	Null
DESCRIPTION	Scottish Gaelic
UPDATE_DATE_TIME	The timestamp at the time of insert.

2. Create a row in the RK_TRANSLATION_DETAIL for every key in the RK_ TRANSLATION_LEY table. The following SQL does this, inserting a null value as the translation for each key (assumes the new locale ID in previous step was 99).

```
insert into rk_translation_detail
       (detail_id, locale_id, key_id, detail_value, update_date_time)
select to_char(rk_translation_detail_seq.nextval),
       99,
        key.key_id,
       null detail_value,
       trunc(date_utils.get_current_gmt())
  from rk_translation_key key;
```

3. Create a translation for each of the records inserted into RK_TRANSLATION_ DETAIL. SIM's PC client contains administrative screens to perform this step. See the *Oracle Retail Store Inventory Management User Guide*.

Assign Correct Locale to User

Update user information so that AC_USER.LANGUAGE and AC_USER.COUNTRY contain the same two character codes (GD & GB) as the new locale that was inserted.

Customizing Barcodes

Barcodes are handled in SIM by barcode processors. The first step of customization is to understand the BARCODE_PROCESSOR table in the database. This table is read to determine which barcode processors should be used when searching for items based on an input string.

Table 7–13 BARCODE_PROCESSOR

Column Name	Data Type	Nullable	Description
NAME	VARCHAR2(25)	No	The name of the format processor.
DESCRIPTION	VARCHAR2(256)	Yes	The description of the format processor.
FILE_NAME	VARCHAR2(256)	No	The full classpath JAVA class name of the processor.
ACTIVE	VARCHAR2(1)	No	**Y** indicates it is active and **N** indicates it is not active.

The NAME and DESCRIPTION rows are not used and are simply there to label the record in the database.

The FILE_NAME row represents the fully qualified classpath to the .java file that implements that particular barcode. For example, the UPC-E barcode processor FILE_ NAME value supplied by SIM is oracle.retail.sim.closed.item.barcode.BarcodeProcessorUPCE.

The ACTIVE row must have a **Y** or **N** value to determine whether or not it is used in the system. The first and easiest customization is to activate or deactivate barcode processors based on usage. SIM provides several barcode processors which are all marked as **Active** by default. Updating the ACTIVE column to **N** for those not used saves processing time.

To add new barcode processors to the system, first code a new barcode processor java class and then insert a row into the BARCODE_PROCESSOR table like in the following example. This example is the processor for a 24-character barcode. The java class must be placed in the classpath of the deployed server.

Table 7–14 24-CHARACTER BARCODE_PROCESSOR

Column Name	Data Type	Nullable	Description
NAME	VARCHAR2(25)	No	UPC-24
DESCRIPTION	VARCHAR2(256)	Yes	24 character barcode processor
FILE_NAME	VARCHAR2(256)	No	custom.barcode.BarcodeProcessorUPC24
ACTIVE	VARCHAR2(1)	No	Y

Creating a Barcode Processor

Every barcode processor must implement the BarcodeProcessor interface:

```
public class BarcodeProcessorUPCE implements BarcodeProcessor
```

There are several methods defined by the interface. The purpose of each of these methods is outlined in the following list. Implement the logic of each method to satisfy the requirements of the barcode.

- `Method: boolean isPossibleFormatMatch(String barcode)`

  This method must return **true** if it is possible that the barcode be a match to this barcode processor format; **false** if it does not.

- `Method: boolean isType2Format();`

  This method must return **true** if the barcode processor is for a type 2 barcode; **false** otherwise.

- `Method: String getItemId(String barcode);`

  This method must parse the item identifier out of the barcode and return this value.

- `Method: int getPrefix(String barcode)`

  The method must return a known prefix to the barcode assuming that the barcode has a standardized prefix (for example, all Type 2 barcodes start with **2**). Return **-1** if no standard prefix exists.

- `Method: String getFormatCode();`

  Returns the format code for the barcode format. This format code is the value that must be assigned to the item in the AS_ITM table in the BARCODE_FORMAT column.

- `Method: Boolean isPriceSupported();`

  Return **true** if the barcode contains a price value; **false** otherwise.

- `Method: String getPrice(String barcode);`

  Returns the price portion of the barcode, or null if no price exists in barcode.

A full example of the BarcodeProcessorUPC24:

```
public class BarcodeProcessorUPCE implements BarcodeProcessor {
    protected int totalLength = 24;
    protected String formatCode = "UPC24";
    public boolean isPossibleFormatMatch(String barcode) {
        if (barcode == null) {
            return false;
        }
        return (barcode.length() == totalLength);
    }
    public boolean isType2Format() {
        return false;
    }
    public String getItemId(String barcode) {
         return barcode.substring(3, 14));
    }
    public int getPrefix(String barcode) {
        return -1;
    }
    public String getFormatCode() {
        return formatCode;
    }
    public boolean isPriceSupported() {
        return true;
    }
    public String getPrice(String barcode) {
         return barcode.substring(16, 22));
    }
}
```

Customizing E-mail Alerts

This section of the document covers e-mail alerts within SIM and how to customize their content.

The E-mail Server Configuration

The e-mail server is determined through the property MAIL_JDNI_NAME in server.cfg configuration file. The value for this property determines the session name on the server that should be used. The following is the default property.

```
MAIL_JNDI_NAME=mail/SimMailSession
```

This jndi name is used to retrieve a Session from the server from which we can get a Transport object and send a MimeMessage object, the fundamental classes of sending e-mails. See Class definitions for further explanation.

The actual mail server associated with mail/SimMailSession is assigned to the server during the build and deploy. The install configuration templates include an input value for this input.mail.host that is populated when the template is loaded by looking in the specific properties files, so the ant.install.properties should contain a value for input.mail.host such as the following:

```
## Properties from Page:MailSessionDetails
input.mail.host = sysserver01
```

E-mail Server Class Definitions

The following is the standard javax package classes that are used as part of the e-mail alert system. For more information, access the documentation for the current version of JAVA used in the system.

javax.mail.Session

The Session class represents a mail session and is not sub-classed. It collects together properties and defaults used by the mail APIs. A single default session can be shared by multiple applications on the desktop. Unshared sessions can also be created. The Session class provides access to the protocol providers that implement the Store, Transport, and related classes. The protocol providers are configured using the following files:

- javamail.providers and javamail.default.providers
- javamail.address.map and javamail.default.address.map

javax.mail.Transport

An abstract class that models a message transport. Subclasses provide actual implementations. Note that Transport extends the Service class, which provides many common methods for naming transports, connecting to transports, and listening to connection events.

javax.mail.Message

This class models an e-mail message. This is an abstract class. Subclasses provide actual implementations. Message implements the Part interface. Message contains a set of attributes and content. Messages within a folder also have a set of flags that describe its state within the folder.

javax.mail.internet.MimeMesage

This class represents a MIME-style e-mail message. It implements the Message abstract class and the MimePart interface. Clients wanting to create new MIME-style messages instantiate an empty MimeMessage object and then fill it with appropriate attributes and content. This is the type of mail message that SIM uses.

SIM E-mail System Configuration Values

The following is the list of system configuration values that can be selected or entered for SIM. This can be done through the system administration area of the client application.

EMAIL_FROM_NAME

When the system sends e-mail alerts, this is the e-mail address that is displayed in the **from** area of the e-mail.

TRANSFER_DAMAGED_EMAIL_ALERT

If set to **yes**, sends an e-mail alert when the receiving store receives goods as damaged.

TRANSFER_DISPATCH_EMAIL_ALERT

If set to **yes**, sends an e-mail alert when the sending store dispatches a transfer.

TRANSFER_OVER_UNDER_EMAIL_ALERT

If set to **yes**, sends an e-mail alert when the receiving store received under/over the transferred quantity.

TRANSFER_REQUEST_APPROVE_EMAIL_ALERT
If set to **yes**, sends an e-mail alert when the transfer request has been approved by the sending store.

TRANSFER_REQUEST_REJECT_EMAIL_ALERT
If set to **yes**, sends an e-mail alert when the transfer request has been rejected by the sending store.

DAYS_TO_HOLD_DISPATCHED_TRANSFER_BEFORE_SENDING_EMAIL_ALERT
The number of days before an e-mail alert about a dispatched transfer is sent.

DAYS_TO_SEND_EMAIL_ALERT_BEFORE_NOT_AFTER_DATE_FOR_RETURN_REQUESTS
Returns requests generated in an external system sometimes require the return to be dispatched to the warehouse before a certain date. This option prompts the recipient of the e-mail the specified number of days before the not-after-date is reached if the return was not dispatched.

SIM E-mail Batch Jobs

There are two batch jobs that can be run that produce e-mail alerts in the system:

TransfersOverdueBatchJob
Batch job that finds dispatched transfers that have not been received and then generates/sends an e-mail to users whom have correct permissions. The alert is only sent if the number of days specified in the Days To Hold Dispatched Transfer Before Alert system configuration option has been exceeded.

ReturnNotAfterDateAlertJob
Batch job that generates and sends e-mail alerts for any Requested or Pending returns with a not-after-date coming up within the number of days specified in the **Days To Send Email Alert Before Not After Date Return Requests** system configuration option.

SIM Developed Classes Involved

The following is a brief summary of the SIM-developed JAVA classes used in the e-mail alert process:

EmailDispatcher.java
This class is responsible for sending e-mail alerts. It validates users and permissions to determine who gets e-mailed and performs the actual sending of the e-mail.

EmailAlert.java
This interface is implemented by all e-mail alert commands. It contains the method APIs necessary for the e-mail framework classes to perform generic processes on e-mail alerts.

EmailAlertCommand
EmailAlertCommands are the java classes that construct the content of the e-mail to send to the users. This is the superclass of all e-mail commands that follows the command pattern and implements the EmailAlert interface. Each sub-class of this class must implement the interface methods of EmailAlert.

ReturnRequestEmailAlertCommand
This command builds the e-mail that notifies users when an active Return is nearing its not-after-date.

TransferDamageEmailAlertCommand
This command builds the e-mail that notifies users when items have been received as damaged.

TransferDispatchRequestEmailAlertCommand
This command builds the e-mail that notifies users when a transfer has been requested.

TransferDispatchedEmailAlertCommand
This command builds the e-mail that notifies users when a transfer has been dispatched.

TransferDispatchedOverdueEmailAlertCommand
This command builds the e-mail that notifies users when a dispatched transfer is overdue.

TransferOverUnderReceiptEmailAlertCommand
This command builds the e-mail that notifies users when a transfer is received with quantities that are different than the amount transferred.

UINStoreAlteredEmailAlertCommand
This command builds the e-mail that notifies users that a UIN has had its store location moved without going through a normal inventory transaction.

UserPasswordAssignmentEmailAlertCommand
This command builds the e-mail that notifies a user when the system has assigned a new password to their account.

Customizing an E-mail

The first step of customizing an e-mail alert is to subclass SIM's e-mail alert and add custom logic. The **TransferDamageEmailAlertCommand** is customized as an example. The easiest way to accomplish this is to sub-class the original command. Every e-mail alert must implement the EmailAlert interface. The interface consists of the following for methods. Customization will only require altering the createSubjectLine() and createEmailContact() method:

- `Method: String getPermissionName();`

  This returns the name of the permission required to receive this e-mail from the system.

- `Method: Long getDestinationId();`

  This method returns the store identifier of the store to receive the e-mail.

- `Method: String createSubjectLine(Locale locale);`

  Returns the subject line of the e-mail. Locale might be used to internationalize the e-mail subject line.

- `Method: String createEmailContent (Locale locale);`

  Returns the content of the e-mail message. Locale may be used to internationalize the e-mail content.

```
public class CustomizedTransferDamageEmailAlert extends
TransferDamageEmailAlertCommand {
    private Transfer transfer;
    public void setTransfer(Transfer transfer) {
        this.setTransfer(transfer);
```

```
        this.transfer = transfer;
    }
    public String createSubjectLine(Locale locale) {
        return TranslatorServerCache.getMessage(locale, "Custom Transfer Damaged
Subject Line");
    }
    public String createEmailContent(Locale locale) {
        StringWriter emailContent = new StringWriter();
        PrintWriter printWriter = new PrintWriter(emailContent);
        printWriter.println(TranslatorServerCache.getMessage(locale,
               "Damaged goods received for transfer: ", transfer.getId()));
        printWriter.println();
        return emailContent.getBuffer().toString();
    }
}
```

The second step is very similar to other customization patterns: the factory implementation is altered to return the customized e-mail alert, and then server.cfg is updated to point at the new implementation. The custom factory implementation should extend the regular factory implementation and then override only the single create() method that it needs to:

```
public class CustomizedCommandFactoryImpl extends CommandFactoryImpl {
    public TransferDamageEmailAlertCommand createTransferDamageEmailAlertCommand()
{
        return new CustomizedTransferDamageEmailAlert();
    }
}
```

To configure the factory, simply modify server.cfg and set the value to the customized class:

```
COMMAND_FACTORY_IMPL=customized.source.CustomizedCommandFactoryImpl
```

8

Internationalization

Internationalization is the process of creating software that can be translated easily. Changes to the code are not specific to any particular market. SIM has been internationalized to support multiple languages.

This section describes configuration settings and features of the software that ensure that the base application can handle multiple languages.

Translation

Translation is the process of interpreting and adapting text from one language into another. Although the code itself is not translated, components of the application that are translated may include the following, among others:

- Graphical user interface (GUI)
- Error messages

The following components are not usually translated:

- Documentation (Online Help, Release Notes, Installation Guide, User Guide, Operations Guide)
- Batch programs and messages
- Log files
- Configuration Tools
- Reports
- Demonstration data
- Training Materials

The user interface for SIM has been translated into:

- Chinese (Simplified)
- Chinese (Traditional)
- Croatian
- Dutch
- French
- German
- Greek
- Hungarian

- Italian
- Japanese
- Korean
- Polish
- Portuguese (Brazilian)
- Russian
- Spanish
- Swedish
- Turkish

DAO Layer

Tables

Three tables exist in the database to support internationalization: RK_TRANSLATION_LOCALE, RK_TRANSLATION_KEY and RK_TRANSLATION_DETAIL. Details about these tables can be found in the SIM Data Model documentation. The last remaining table used in the process is the AC_USER table. The LANGUAGE and COUNTRY columns are used to determine the country and language of the employee that logs in. Alternatively, the locale information for an employee can be filled in from an LDAP connection. The employee's locale is matched with the RK_TRANSLATION_LOCALE table to retrieve translation information.

Loading Data

Data load scripts populate the RK_TRANSLATION_LOCALE table on installation. Upon data load, one record is created in the RK_TRANSLATION_DETAIL table for each key in the RK_TRANSLATION_KEY table paired with each locale id in the RK_TRANSLATION_LOCALE table.

Retrieving Translations

When retrieving translations, the displayable text is first retrieved from the RK_TRANSLATION_DETAIL table for the locale and key involved. If this value is missing or the DETAIL_VALUE column is empty, then the KEY value is returned from the RK_TRANSLATION_KEY table as both the key and the value.

When retrieving translations for a locale, the detail value is read from the RK_TRANSLATION_DETAIL table for the language (with country and variant set to null). If a country exists in the locale, the country information is read from the DETAIL table and its values replace those read for the language. If a variant exists, the variant information is read from the DETAIL table and its values replace those of the language and country. Of course, this only occurs if an actual translation is found at the country and variant level (it will not suddenly null out the language value).

Types of Internationalization

Logging

Service layer error message logging does not attempt to translate the information at this time.

Rules

The string parameter passed into the RulesInfo constructor within a Rule class is the language key used for translation.

Example 8–1 ItemMustBeSellableRule

```
public final class ItemMustBeSellableRule extends SimRule {
private static final RulesInfo RULE_FAILED = new RulesInfo("Item not sellable.");
```

PC UI Labels and Titles

Everywhere within the entire SIM Swing framework that a label or title is used, the translation takes place automatically within the component. All title and label strings are the keys into the translation functionality. Failure to find a translation for the label or title simply returns the label or title itself.

Example 8–2 ItemLookupPanel

```
private RTextFieldEditor itemDescriptionEditor = new RTextFieldEditor("Item
Description");
private RIntegerFieldEditor searchLimitEditor = new RIntegerFieldEditor("Item
Description");
```

Error Messages and Exception

Error messages are long explanations of some validation or event failure that occurred in the system. The text message within the exception is the key into the translation functionality.

When dealing with error messages on the server, there should be no attempt at formatting or translation. Formatting and translation are strictly a client display responsibility. Simply create a business exception assigning the correct message and parameters and throw the error message to the client.

Dynamic Value Messages in Exceptions

BusinessExceptions are capable of handling dynamic values internally. The constructor that takes parameters uses these parameters to complete the dynamic message string. A very good example is found in the LocationSequencer class when re-sequencing allocations. The max location error takes two dynamic numbers. The following example shows how those numbers are placed into a parameter array and passed in the construction of a business exception.

Example 8–3 LocationSequencer

```
public class LocationSequencer {

    public void resequenceSectionOfLocations(List values, int startingIndex, long
```

```
startGap) throws Exception {
        // if size is greater then max values - then we have an error
        if (values.size() > maxOrderValue) {
            Object[] params = new Object[2];
            params[0] = getMaxOrderValue();
            params[1] = Integer.valueOf(values.size());
            throw new BusinessException(ErrorKey.SEQUENCE_MAX_LOCATION_ERROR,
params);
        }
 // Additional Code
}
```

Dates

No work needs be done by the developer to internationalize dates within the application. Both RDateField and RDateFieldEditor handle all of the logic, the developer simply needs to use a java.util.Date object. All conversion from text to Date and Date to text is handled by these editors. See how setDate() and getDate() are called on the editor in the following example. The date and calendar are displayed in the language and style of the locale set for the user who has logged on.

Example 8–4 InventoryAdjustmentFilterDialog

```
private RDateFieldEditor fromDateEditor = new RDateFieldEditor("From Date");
private RDateFieldEditor toDateEditor = new RDateFieldEditor("To Date");

    public void setFilter(InventoryAdjustmentQueryFilter filter) throws Exception
{
        model.setFilter(filter);

        fromDateEditor.setDate(filter.getFromDate());
        toDateEditor.setDate(filter.getToDate());
// Additional Code
  }

    private void doSearch() throws Exception {
        InventoryAdjustmentQueryFilter filter = model.getFilter();

filter.setDateRange(fromDateEditor.getDateAtStartOfDay(),
toDateEditor.getDateAtEndOfDay());
       filter.setInventoryAdjustmentId(adjustmentEditor.getLongOrNull());
// Additional Code
}
    }
```

When formatting your own dates on PC Client (not through an editor), use the LocaleManager object, which has several methods for formatting dates. This automatically uses the currently assigned locale.

Another way to format dates is by using the DateMask, which allows the developer to assign a format type before formatting the date. DateMask can be used on any visual component that takes a mask. DateDisplayer formats a Date only in the SHORT format, but can used on any visual component that takes a displayer.

There are four dates allowed within the system following a standard java convention: SHORT, MEDIUM, LONG and FULL. All RDateFieldEditors within the application are assigned the SHORT format. In addition, the format values for each of the date formats will be standard JAVA format sequences by default. These default values can be overridden in the Date.cfg file by defining the JAVA format sequence to use for the date format type.

The Date.cfg file keys begin with the 2-character language and country code for the locale to be formatted. This is followed by the .dateType to be formatted. The format value must be in standard java convention. firstDayOfWeek determines the first day of week to be displayed on calendars. wirelessInput is the entry parser of handheld date entry fields. wirelessOutput is the date formatting of handheld date entry fields. wirelessDisplay is for handheld date display only.

Example 8–5 Date.cfg

```
# ENGLISH - AUSTRALIAN
#enAU.firstDayOfWeek=1
#enAU.entryDate=d/MM/yy
#enAU.shortDate=d/MM/yy
#enAU.mediumDate=d/MM/yyyy
#enAU.longDate=d MMMM yyyy
#enAU.fullDate=EEEE, d MMMM yyyy
enAU.monthPattern=MM-dd
enAU.wirelessInput=dd-MM-yy,ddMMyy
enAU.wirelessOutput=dd-MM-yy
enAU.wirelessDisplay=dd-mm-yy
enAU
```

Money

SimMoney is the data object that represents money within the system. Currency is a standard JAVA object that represents the type of money being represented. A Money object consists of a BigDecimal amount and a String currencyCode (though only Currency can set in the constructor). This is because once a money object is created, changing its currency would invalidate any amounts it represented. SIM does not handle currency conversion.

SimMoney is very similar to Date in that the RMoneyFieldEditor handles all of the internationalization for the user. RMoneyFieldEditor edits a Locale, Currency and BigDecimal in a generic fashion. The SimMoneyFieldEditor is a subclass that edits the SimMoney field directly. The following example demonstrates using the SimMoneyFieldEditor.

Note: Currency is handled separately from Locale by the editor. Currency describes the type of money being displayed while Locale indicates the desired language display format for the currency.

Example 8–6 ItemTicketDetailPanel

```
    private SimMoneyFieldEditor overridePriceEditor = new
SimMoneyFieldEditor("Override Ticket Price");
   private void loadEditorInformation(ItemTicket itemTicket) throws Exception {

            overridePriceEditor.setMoney(itemTicket.getOverridePrice());

// Code To load rest of information
}

    private void doPriceModified() {
        try {

model.getItemTicket().setOverridePrice(overridePriceEditor.getMoney());
        } catch (BusinessException buException) {
            // Deal With Exceptions
        }
    }
```

Wireless Internationalization

Internationalization is handled by different approaches based on where in the Wireless code the translation is needed.

Forms

In the xml files that define the handheld forms, the display of labels is usually surrounded with a $[ ] indicator. In the Form<name>.java classes, the text within the brackets is wrapped by an AppGlobal.getString() call which translates the text.

Example 8–7 Screen_ContainerLookupScreen.xml

```
<Screen name="ContainerLookupDetail">
  <LogicalScreen>
    <field name="containerId" type="string" length="21"/>
    // More Field Names…
    <field name="asnNumber" type="string" length="21"/>
  </LogicalScreen>

  <PhysicalScreens deviceclass="dnw">
    <PhysicalScreen seq="0">
    <label y="0" x="0" width="21" height="1" style=".heading1">
         $[Lookup Results]</label>
    <label y="2" x="0" width="11" height="1" >
         $[Container]$[wireless.delimiter]</label>
 <label y="2" x="11" width="21" height="1" name="containerId"
         field="containerId"/>
 <label y="3" x="0" width="8" height="1" >
         $[Status]$[wireless.delimiter]</label>
     // More labels…
     <label y="12" x="13" width="21" height="1" name="totalCases"
         field="totalCases" />

     <cmdkey key="&exit;" y="16" x="0" height = "0" width="0"
         name="return" action="callMethod" target="doExit"/>
  </PhysicalScreen>
  </PhysicalScreens>
</Screen>
```

Example 8–8 Form_dnw_ContainerLookupDetail_0.java

```
 public Form_dnw_ContainerLookupDetail_0(String id, EventHandler_
ContainerLookupDetail handler)
   throws WaveLinkError {
  super(id, false, handler);

  add(new RFPrintLabel(wlio, AppGlobal.getString("Container") +
      AppGlobal.getString("wireless.delimiter"), 0, 2, 11, 1, termWidth));
  add(new RFPrintLabel(wlio, AppGlobal.getString("Status") +
      AppGlobal.getString("wireless.delimiter"), 0, 3, 8, 1, termWidth));
```

EventHandlers

Every event handler extends from SimEventHandler, which contains numerous methods to assist in formatting and retrieving information in an internationalized manner. These helper methods in the superclass should always be used to perform these types of tasks when coding event handlers.

Key methods include:

SetFormDate()
Formats a date for the locale and country and places it in the form.

SetFormInteger()
Formats an integer for the locale and places it in the form.

SetFormDecimal()
Formats a decimal for the locale and places it in the form.

SetFormQuantity()
Formats a quantity for the locale and places it in the form.

GetText()
Retrieves the translation of the text.

GetLabel()
Retrieves the translation of the text followed by the label delimiter

GetMessage()
Retrieves the translation of the message

GetFormInteger()
Retrieves entered text as an integer

GetFormDecimal()
Retrieves entered text as a decimal

GetFormQuantity()
Retrieves entered text as a quantity

HandleException()
Handles displaying an exception (translating the message)

Here are some examples of standard eventhandler code using these internationalization methods. In the first example, we are translating the label **Select PO From**.

Example 8–9 EventHandler_DirectDeliverySelectPO

```
protected void onFormOpen() {
  try {
    setFormData(FIELD_INSTRUCTIONS, getLabel("Select PO From"));
    setFormData(FIELD_SUPPLIER,
        DirectDeliveryWirelessUtility.getContext().getSource().getName());
    // Some Code
  } catch (Exception e) {
    handleException(e);
  }
}
```

Example 8–10 EventHandler_ContainerLookupDetail

```
EventHandler_ContainerLookupDetail
    protected void onFormOpen() {
        try {
            LogService.debug(this, this.getClass().getSimpleName() +
".onFormOpen()");
            ShipmentCartonVO cartonVO = (ShipmentCartonVO)
SimWirelessRepository.getValue(ContainerWirelessKeys.USER_CONTEXT_CONTAINER);
            SourceVO fromLocation = cartonVO.getFromLocation();

            setFormText("containerId", cartonVO.getCartonId());
            setFormText("status", getText(cartonVO.getStatus().toString()));
            setFormText("fromLocation", fromLocation.getId() + " " +
fromLocation.getName());
            setFormText("asnNumber", cartonVO.getAsnId());
            setFormDate("eta", cartonVO.getEta());
            setFormDate("receiptDate", cartonVO.getReceiveDate());
            setFormText("receiptTime",
LocaleWirelessUtility.formatTime(cartonVO.getReceiveDate()));
            setFormInteger("totalCases", cartonVO.getNumberOfCasesExpected());
        } catch (Exception e) {
            handleException(e);
        }
    }
```

<name>WirelessUtility

Every wireless utility class extends from WirelessUtility, which like SimEventHandler, contains numerous methods to assist in formatting and retrieving information in an internationalized manner. These helper methods in the superclass should always be used to perform these types of tasks when coding utility methods.

Key methods include:

GetText()
Retrieves the translation of the text.

Alert()
Handles displaying an alert message (translating the message)

GetLabel()
Retrieves the translation of the text followed by the label delimiter

GetMessage()
Retrieves the translation of the message

HandleException()

Handles displaying an exception (translating the message)

Here is an example of standard utility code using these internationalization methods.

Example 8–11 StockCountWirelessUtility

```
public static String getDescription(StockCount stockCount) {
    StringBuilder buffer = new StringBuilder();
    try {
        if (stockCount.getType() == StockCountType.PROBLEM_LINE) {
            buffer.append(getLabel("wireless.problemLineABBV"));
        }
        String text = stockCount.getCountDescription().trim();
        int index = text.lastIndexOf("(");
        if (index < 0) {
            buffer.append(text);
        } else {
            buffer.append(text.substring(index));
            buffer.append(StringConstants.SPACE);
            buffer.append(text.substring(0, index).trim());
        }
        return buffer.toString();
    } catch (Exception e) {
        return getText("wireless.noDescABBV");
    }
}
```

In the previous example, business logic is required to create a stock count description. The utility uses the getLabel() and getText() helper methods to guarantee that the text is translated as the description is being built.

LocaleWirelessUtility

If there are no superclass helper methods available for what you want to accomplish, you can directly use the LocaleWirelessUtility to perform internationalized functions.

Wireless Labels

Wireless labels have an additional consideration that is not needed on the PC. The width of a wireless form, which is displayed on a handheld device, is very narrow. That means only a small amount of space is allocated to a label. The English labels used as translation keys are defined by the space they take up. Oracle Retail suggests that instead of using a standard English label such as **status** for the key, use **wireless.status** instead, so that it is clear in the GUI administrative screen that the translation being supplied is for the wireless device.

Handheld Device Configuration for Japanese Display

This white paper explains how to configure the Wavelink Client to display Japanese text.

The following document is available through My Oracle Support (formerly MetaLink). Access My Oracle Support at the following URL:

https://support.oracle.com

Oracle Retail White Paper: Oracle Retail Store Inventory Management Handheld Device Configuration for Japanese Display **(Doc ID: 601817.1)**

Customizing Internationalization

The PC client provides administrative screens for adding and altering translations.

Customizing a Language

The administrative section of the PC user interface has the ability to customize the translations of any language provided in the release of SIM. Users may even create completely new keys not previously existing in SIM. An understanding of english is necessary to use this feature of SIM. See the *Oracle Retail Store Inventory Management Users Guide* for more information about how to use this section of our administrative features.

Do the following to add a new language to SIM:

1. Insert a record into RK_TRANSLATION_LOCALE.
2. Create RK_TRANSLATION_DETAIL row.
3. Create translation.
4. Assign correct locale to user.

Insert a Record Into RK_TRANSLATION_LOCALE

For example, if Scottish Gaelic was to be added as a new language, insert the following row into the database. The LANGUAGE is the two character ISO 639 code for the language. The COUNTRY is the ISO 3166 two-character country code. VARIANT is not required:

Table 8–1 RK_TRANSLATION_LOCALE Record

Column	Value
LOCALE_ID	Insert a new unique ID here.
LANGUAGE	GD
COUNTRY	GB
VARIANT	Null
DESCRIPTION	Scottish Gaelic
UPDATE_DATE_TIME	The timestamp at the time of insert.

Create RK_TRANSLATION_DETAIL Row

The second step is to create a row in the RK_TRANSLATION_DETAIL for every key in the RK_TRANSLATION_LEY table. The following SQL inserts a null value as the translation for each key (assumes the new locale ID in previous step was 99):

```
insert into rk_translation_detail
       (detail_id, locale_id, key_id, detail_value, update_date_time)
select to_char(rk_translation_detail_seq.nextval),
       99,
       key.key_id,
       null detail_value,
       trunc(date_utils.get_current_gmt())
from rk_translation_key key;
```

Create Translation

Create a translation for each of the records inserted into RK_TRANSLATION_DETAIL. The SIM PC client contains administrative screens to perform this step. See the *Oracle Retail Store Inventory Management Users Guide*.

Assign Correct Locale To User

Update user information so that AC_USER.LANGUAGE and AC_USER.COUNTRY contain the same two character codes (GD & GB) as the new locale that was inserted.

Brazil-Specific Setup

The business process selected to support SIM in Brazil is based on the logic that no receipt can take place until a Nota Fiscal (NF) has been confirmed at the receiving store. A Nota Fiscal document is similar to an invoice or bill of lading (BOL), but specific to Brazil. It contains quantities, cost, taxes, to and from location information.

SIM does not have specific Brazil indicators, but rather many specialized system options and security permissions allowing for a more flexible deployment. These indicators need to be set in the suggested configuration or ORFM might not function correctly.

Direct Store Delivery

DSDs in Brazil are not allowed to be started until a Nota Fiscal has been entered. Also, a Nota Fiscal cannot be entered until a PO is created. As such, users should be set up to not allow DSD PO creation nor should they be able to create ASNs in SIM.

If any quantities are added above the expected receipt, the user should remove the physical extra quantities or unexpected items when confirming. The user must scan the extra quantities, and SIM will prompt the user that they should be removed. The unexpected items or overage quantities will be published separately from the expected quantities so ORFM can generate the necessary return documentation.

The following store/system parameters should be set:

- Auto Remove DSD over-receiving – **True**
- Auto remove DSD damaged items – **True**

The following security options for both PC and HH should not be granted:

- Create Direct Delivery
- Review Direct Delivery
- Create new Purchase Order
- Create new ASN

Internal Deliveries

Warehouse Deliveries and Store to Store Transfer are not allowed to have any discrepancies from the Nota Fiscal. To enforce this process SIM will auto receive the warehouse and transfer deliveries coming from ORFM. This means that no detailed receiving is allowed.

After SIM receives the ASN from the warehouse, or the other store, SIM will get a second ASN notice that will trigger the auto receiving process. This means that users should be prevented from adjusting the transfer of warehouse delivery.

The following store parameters should be set:

- Warehouse Auto Receive – External message
- Store Auto Receive – External message

In addition, all stores should be set up for auto-receiving.

Since no receiving is allowed for warehouse deliveries, only the following security privileges should be granted:

- Access Warehouse Delivery – This enables the user to view the warehouse deliveries, but not make any changes. No other Warehouse delivery privileges should be granted.

Since no receiving is allowed for store deliveries, the following transfer security privileges should not be granted:

- Receive all items on transfer
- Receive all on transfer
- Receive item on transfer
- Complete transfer Receipt
- Receive transfer
- Add item to transfer receipt
- Complete transfer receipt

Receiver Unit Adjustments

Receiver unit adjustments are not allowed in Brazil, so the following system options need to be set:

- Number of days Received Transfers can be adjusted – **0**
- Number of days received warehouse deliveries can be adjusted – **0**
- Number of days Direct Deliveries can be adjusted – **0**

Unsupported Processes

Vendor ASNs

Vendor-created ASNs are not supported, since the only valid receipts that can be made are against the NF.

Serialization

Because detailed receiving is not supported in Brazil, it is not possible to register or use UINs.

A

Appendix: SIM Permissions

The following table describes the permissions supported by SIM.

Table A–1 SIM Permissions

Permission	Type	Topic	Usage
Access Admin	PC	Admin	With this permission, the **Admin** button on the SIM Login screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Auto-Receive Stores	PC	Admin	With this permission, the **Auto-Receive Stores** button on the Store Admin screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Buddy Stores	PC	Admin	With this permission, the **Buddy Store** button on the Store Admin screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Container Lookups	Handheld	Admin	With this permission the Container Lookup menu option on the Lookups Menu will be displayed. Without this permission the menu option will not be displayed.
Access Container Lookups	PC	Admin	With this permission, the **Container Lookup** button on the Lookups Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Customer Order	PC	Admin	With this permission, the **Customer Orders** button on the Lookups Screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, when a user double clicks on a Customer Order Record from the Customer Order tab in Item Details, the Customer Orders Detail Screen will open. Without this permission, the user will not be taken to the Customer Order Detail screen and will remain on the Item Details Screen.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Access Customize Translations	PC	Admin	With this permission the **Customize Translations** button on the UI Config screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Finisher Lookups	PC	Admin	With this privilege, the Finisher Lookup button on the Lookups Screen will be displayed and enabled. Without this privilege the button will not be displayed.
Access Formats	PC	Admin	With this permission, the **Formats** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Inventory Adjustment Reasons	PC	Admin	With this permission the **Inventory Adjustment Reason** button on the Setup screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Inventory Management	Handheld	Admin	With this permission the Inv. Management menu option on the Main Menu will be displayed. Without this permission the menu option will not be displayed.
Access Inventory Management	PC	Admin	With this permission, the **Inv Mgmt** button on the SIM Login Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Item Lookups	Handheld	Admin	With this permission the Item Lookups menu option on the Lookups Menu will be displayed. Without this permission the menu option will not be displayed.
Access Item Lookups	PC	Admin	With this permission, the **Item Lookup** button on the Lookups Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Lookups	Handheld	Admin	With this permission the Lookups menu option on the Main Menu will be displayed. Without this permission the menu option will not be displayed.
Access Lookups	PC	Admin	With this permission, the **Lookups** button on the SIM Login Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Polling Timers	PC	Admin	With this permission the **Polling Timers** button on the Technical Maintenance screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Access Printers	PC	Admin	With this permission, the **Printers** button on the Admin>Setup screen will be displayed and enabled. Without this permission, the tab will not be displayed.
Access Printer Selections	PC	Admin	With this permission, the **Printer Selections** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Product Groups	PC	Admin	With this permission, the **Product Group** button on the Admin Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Product Group Schedules	PC	Admin	With this permission, the **Product Group Schedule** button on the Admin Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Reports	PC	Admin	With this permission, the **Reports** button on the SIM Login Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Setup	PC	Admin	With this permission the **Setup** button on the Admin screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Shipping and Receiving	Handheld	Admin	With this permission the Shipping/Receiving menu option on the Main Menu will be displayed. Without this permission the menu option will not be displayed.
Access Shipping and Receiving	PC	Admin	With this permission, the **Shipping/Receiving** button on the SIM Login Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access SIM Managed Stores	PC	Admin	With this permission, the **SIM Managed Stores** button on the Store Admin screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access SIM Stores	PC	Admin	With this permission, the **SIM Stores** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Access Store Default Admin	PC	Admin	With this permission, the **Store Default Admin** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Supplier Lookups	Handheld	Admin	With this permission the Supplier Lookup menu option on the Lookups Menu will be displayed. Without this permission the menu option will not be displayed.
Access Supplier Lookups	PC	Admin	With this permission, the **Supplier Lookup** button on the Lookups Screen will be displayed and enabled.It also is required to display the Primary Supplier button on the Item Detail screen. Without this permission, the button will not be displayed.
Access Staged Messages	PC	Admin	With this permission the **Staged Messages** button on the Technical Maintenance screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Store Admin	PC	Admin	With this permission, the **Store Admin** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access System Admin	PC	Admin	With this permission, the **System Admin** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Technical Maintenance	PC	Admin	With this permission the **Technical Maintenance** button on the Admin screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access UDAs	Handheld	Admin	With this permission the UDA rmenu option is enabled on the Item Lookup menu. Without this privilege the user cannot see any UDAs.
Access UDAs	PC	Admin	Allows the UDA option button to be enabled on the Item Lookup screen, enable the UDA printer/label screen, and enable the UDA button in Item Detail. Without this privilege the user cannot see any UDAs.
Access UI Configuration	PC	Admin	With this permission the **UI Configuration** button on the Technical Maintenance screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Activate Debugging in Client Log	PC	Admin	With this permission, the Repository tab on the Client Status dialog will be displayed, and the debug-activated check box on the Stats tab will be enabled. Without this permission, the repository tab will not be displayed and the debug-activated check box will be disabled.
Add Inventory Adjustment Reason	PC	Admin	With this permission the **Add** button on the Inventory Adjustment Reason screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Product Groups	PC	Admin	With this permission, the **Create** button on the Product Group List screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Product Group Schedules	PC	Admin	With this permission, the **Create** button on the Product Group Schedule List screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Translations	PC	Admin	With this permission the **Create** button on the Translation Details screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete Inventory Adjustment Reason	PC	Admin	PCAdminWith this permission the **Delete** button on the Inventory Adjustment Reason screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete Product Groups	PC	Admin	With this permission, the **Delete** button on the Product Group List screen with be displayed and enabled. Without this permission, the button will not be displayed. If the button is displayed, the user must also have the necessary data permission for the product group the user is attempting to delete. If the user is not authorized for the product group type, **User is not authorized to delete this type of Product Group.** is displayed.
Delete Product Group Schedules	PC	Admin	With this permission, the **Delete** button on the Product Group Schedule List screen will be displayed and enabled. Without this permission, the button will not be displayed. If the button is displayed, the user must also have the necessary data permission for the product group that is associated to the product group schedule that is attempted to be deleted. If the user is not authorized for the product group type, **User is not authorized to delete this type of Product Group Schedule.** is displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delete Staged Messages	PC	Admin	With this permission the **Delete** button on the Staged Messages Lookup screen will be displayed and enabled. Without this permission, the button will not be displayed.
Display Stock Locator	Handheld	Admin	With this privilege, the Stock Locator option on the Item Detail screen will be listed. Selecting this option will navigate to Stock Locator Screen. Without this privilege the option will not be displayed.
Display Stock Locator	PC	Admin	With this permission, the Stock Locator button on the Item Detail screen will be displayed and enabled. Clicking the button will navigate to Stock Locator screen. Without this privilege the button will not be displayed.
Edit Product Groups	PC	Admin	With this permission, when a user double-clicks on an existing Product Group, the Product Group Detail screen will open. If the user also has the correct data permission for the product group type, the screen will open in Edit mode. Without the necessary data permission for the type, the screen will open in View-only mode. The user must also have this permission for each store that is included on the product group. If the user does, then the user can edit the product group; if the user does not, then the screen will open in View-only mode.
Edit Product Group Schedules	PC	Admin	With this permission, when a user double-clicks on an existing Product Group Schedule, the Product Group Schedule Detail screen will open. If the user also has the correct data permission for the product group type, the screen will open in Edit mode. Without the necessary data permission for the type, the screen will open in View only mode. The user must also have this permission for each store that is included on the schedule. If the user does, then the user can edit the schedule; if the user does not then the screen will open in View-only mode.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Access Customer Orders	PC	Customer Orders	With this permission the **Customer Orders** button on the Item Lookup screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission the Customer Order Tab on the Item Lookup popup search screen will be displayed and enabled. Without this permission the tab will be disabled. With this permission the **Customer Orders** button on the Lookups screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Direct Delivery	Handheld	Direct Delivery	With this permission the Direct Delivery menu option on the Shipping/Receiving Screen will be displayed. Without this permission the menu option will not be displayed.
Access Direct Delivery	PC	Direct Delivery	With this permission, the **Direct Delivery** button on the Shipping/Receiving Screen will be displayed and enable. Without this permission, the button will not be displayed.
Access Reject Delivery	Handheld	Direct Delivery	With this permission the Reject Delivery menu option on the Direct Delivery Detail screen will be displayed. Without this permission the menu option will not be displayed.
Access Reject Delivery	PC	Direct Delivery	With this permission, the **Reject Delivery** button on the Direct Delivery Detail screen will be displayed and enable. Without this permission, the button will not be displayed.
Add or Edit Items for Direct Delivery	Handheld	Direct Delivery	Enables a user to add or edit items on the direct delivery. Without this privilege the user cannot edit or add items on the Handheld. The menu option is not available.
Add Item to Direct Delivery	PC	Direct Delivery	With this permission and the Edit Direct Delivery permission, the **Add Item** button on the Direct Delivery Detail Screen will be displayed and enabled when editing an existing Direct Delivery. Without this permission, the button will not be displayed when editing an existing Direct Delivery. This permission does not apply when creating a new direct delivery. The **Add Item** button will always be available when creating a direct delivery.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Allow Damages for Direct Delivery	Handheld	Direct Delivery	Enables a user to add damaged items for a DSD. Without this permission, the damaged menu option on the Handheld is not displayed. On the discrepant flow they can be displayed but are not editable.
Allow Damages for Direct Delivery	PC	Direct Delivery	Enables a user to add or not damaged items for a DSD. This feature displays the column on the DSD detail screen but not allow it to be edited.
Allow Over-receiving for Direct Delivery	PC/Handheld	Direct Delivery	Enables a user to over-receive quantities for a DSD. Will prompt the use when damaged and received quantity is larger than the expected quantity if they do not have this permission.
Complete Direct Delivery	Handheld	Direct Delivery	With this permission the Complete Order menu option on the Direct Delivery Screen will be displayed. Without this permission the menu option will not be displayed.
Confirm Direct Delivery	PC	Direct Delivery	With this permission, the **Confirm** button on the Direct Delivery Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Direct Delivery	Handheld	Direct Delivery	This privilege enables the user to create DSD on the Handheld.
Create Direct Delivery	PC	Direct Delivery	With this permission, the **Create** button on the Direct Delivery List Screen will be displayed and enabled. The **Add Items** button will also be enabled on the Direct Delivery Detail screen. Without this permission, the button will not be displayed.
Create New ASN	PC/Handheld	Direct Delivery	Enables a user to create a new delivery against an existing order. Without this privilege the user cannot create a new delivery on the PC or Handheld.
Create New Purchase Order	Handheld	Direct Delivery	With this permission the Yes menu option on the message asking about creating a new PO will be displayed. Without this permission the Yes menu option will not be displayed and the user will automatically be brought forward in the dialog with the **No** option assumed to be clicked.
Create New Purchase Order	PC	Direct Delivery	Enables a user to create a new direct store delivery on the direct delivery identify screen. The functionality works similar to a non-DSD supplier. Without this privilege the user cannot create a DSD on the PC.
Delete Direct Delivery	Handheld	Direct Delivery	With this permission the Delete Delivery menu option on the Direct Delivery Screen will be displayed. Without this permission the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delete Item from Direct Delivery	PC	Direct Delivery	With this permission, the **Delete** button on the Direct Delivery Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Direct Delivery	PC	Direct Delivery	With this permission, when the user double-clicks on an existing Direct Delivery in the Direct Delivery List Screen, the Direct Delivery Detail screen will open with the Direct Delivery and allow the user to make changes. Without this permission, when the user double-clicks on an existing Direct Delivery, the Direct Delivery Detail screen will open and the user will only be allowed to view the information and not make any changes.
Override Supplier Discrepancies	PC	Direct Delivery	With this permission, the user can override the supplier discrepancies check during the direct delivery receipt.
Override Supplier Discrepancies	Handheld	Direct Delivery	With this permission, the user can override the supplier discrepancies check during the direct delivery receipt.
Receive All for Direct Delivery	PC	Direct Delivery	With this permission, the **Receive All** button on the Direct Delivery Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Record Damages for Direct Delivery	Handheld	Direct Delivery	With this permission the Record Damages menu option on the Direct Delivery Screen will be displayed. Without this permission the menu option will not be displayed.
Review Direct Delivery	Handheld	Direct Delivery	With this permission the Amend Order menu option on the Direct Delivery Screen will be displayed. Without this permission the menu option will not be displayed.
Email Alert –Return Expiration Date Approaching	System	E-mail	When this permission is assigned, the user will be notified if the return expiration date is approaching.
Email Alert – Transfer Damaged Items	System	E-mail	When this permission is assigned, the user will get any e-mail notifications when damaged items are received for a transfer.
Email Alert – Transfer Dispatched	System	E-mail	When this permission is assigned, the user will get any e-mail notifications when a transfer is dispatched.
Email Alert – Transfer Over/Under Received	System	E-mail	When this permission is assigned, the user will get any e-mail notifications when a transfer has over/under received values.
Email Alert – Transfer Request Approved	System	E-mail	When this permission is assigned, the requesting user will get any e-mail notifications when a user in another store approves the request.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Email Alert – Transfer Request Rejected	System	E-mail	When this permission is assigned, the requesting user will get any e-mail notifications when a user in another store rejects the request.
Email Alert for Unexpected UIN	System	E-mail	Notification e-mails are sent when UINs are discovered at a store where they should not be. This permission is checked by the alert processing to determine if the user can be e-mailed this nofication.
Access Inventory Adjustments	Handheld	Inventory Adjustments	With this permission the Inventory Adjustments menu option on the Inv. Management Menu will be displayed. Without this permission the menu option will not be displayed.
Access Inventory Adjustments	PC	Inventory Adjustments	With this permission, the **Inventory Adjustment** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Inventory Adjustments	PC	Inventory Adjustments	With this permission, the **Create** button on the Inventory Adjustment List screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Inventory Adjustments	PC	Inventory Adjustments	With this permission, when the user double-clicks on an existing Inventory Adjustment in the Inventory Adjustment List screen, the Inventory Adjustment Detail screen will open with the Inventory Adjustment and allow the user to make changes. This will also allow for the **Add Item** button to be enabled. Without this permission, when the user double-clicks on an existing Inventory Adjustment, the Inventory Adjustment Detail screen will open and the user will only be allowed to view the information and not make any changes.
Access Item Basket	Handheld	Item Basket	With this privilege the Item Basket menu option on the Inventory Management menu will be displayed. Without this privilege the menu option will not be displayed.
Add Item to Item Basket	Handheld	Item Basket	With this privilege the Add Item option on the Item Basket summary menu will be displayed. Without this privilege the menu option will not be displayed.
Create Item Basket	Handheld	Item Basket	With this privilege the Create Item Basket menu option on the Inv. Management menu will be displayed. Without this privilege the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delete Item Basket	Handheld	Item Basket	With this privilege the Delete Item Basket option on the Item Basket summary menu will be displayed. Without this privilege the menu option will not be displayed.
Delete Item from Item Basket	Handheld	Item Basket	With this privilege the Delete Item option on the Item Basket summary menu will be displayed. Without this privilege the menu option will not be displayed.
Print Item Basket	Handheld	Item Basket	With this privilege the Print Ticket option on the Item Basket summary will be displayed. Without this privilege the menu option will not be displayed.
Save Item Basket	Handheld	Item Basket	Item BasketWith this privilege the Save Basket option on the Item Basket summary will be displayed. Without this privilege the menu option will not be displayed.
View Item Basket	Handheld	Item Basket	With this privilege the View Details option on the Item Basket summary will be displayed. Without this privilege the menu option will not be displayed.
Access Item Requests	Handheld	Item Requests	With this permission the Item Requests menu option on the Inv. Management Menu will be displayed. Without this permission the menu option will not be displayed.
Access Item Requests	PC	Item Requests	With this permission, the **Item Request** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Items to Item Request	PC	Item Requests	With this permission and the Edit Item Requests permission, the **Add Item** button on the Item Request Detail Screen will be displayed and enabled when editing an existing Item Request. Without this permission, the button will not be displayed. This permission does not apply when creating an Item Request. **Add Item** button will always be available when creating.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Add or Edit Item For Item Request	Handheld	Item Requests	With this permission and the Edit Item Request (handheld) permission the Add/Edit Item menu option on the Item Requests Summary screen will be displayed. Without this permission the menu option will not be displayed when editing an Item Request. When editing an Item Request, if the user scans an item that is not on the Item Request, the system will prompt with a message asking if the user wishes to add the item. With this permission, both the Yes and No options should be present. Without this permission the message screen (asking to add item) will not be displayed and the user will automatically be brought forward in the dialog with the **No** option assumed to be clicked. This permission does not apply when creating an item request. Users will always be able to add and edit items when creating an item request.
Create Item Requests	Handheld	Item Requests	With this permission the Create Item Request menu option on the Item Requests Menu will be displayed. Without this permission the menu option will not be displayed.
Create Item Requests	PC	Item Requests	With this permission, the **Create** button on the Item Request List Screen will be displayed and enabled. The **Add Items** button will also be enabled on the Item Request Detail screen. Without this permission, the button will not be displayed.
Delete Item Request s	Handheld	Item Requests	With this permission the Delete Item Request menu option on the Item Requests Menu will be displayed. Without this permission the menu option will not be displayed.
Delete Item Requests	PC	Item Requests	With this permission, the **Delete** button on the Item Request List Screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, the **Delete** button on the Item Request Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delivery Slot Description 1	PC	Item Request	With this privilege, Delivery Slot Description 1 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description does not display in the timeslot dropdowns. ■ Request Delivery Timeslot – header default, dropdown ■ Timeslot – line item column, dropdown ■ Delivery Timeslot – filter dialog dropdown
Delivery Slot Description 1	HH	Item Request	With this privilege, Delivery Slot Description 1 displays in timeslot options on Item Request screens. Without this privilege, this delivery slot description does not display on the timeslot screens.
Delivery Slot Description 2	PC	Item Request	With this privilege, Delivery Slot Description 2 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description will not display in the timeslot dropdowns.
Delivery Slot Description 2	HH	Item Request	With this privilege, Delivery Slot Description 2 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description does not display in the timeslot dropdowns.
Delivery Slot Description 3	PC	Item Request	With this privilege, Delivery Slot Description 3 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description does not display in the timeslot dropdowns.
Delivery Slot Description 3	HH	Item Request	With this privilege, Delivery Slot Description 3 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description does not display in the timeslot dropdowns.
Delivery Slot Description 4	PC	Item Request	With this privilege, Delivery Slot Description 4 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description does not display in the timeslot dropdowns.
Delivery Slot Description 4	HH	Item Request	With this privilege, Delivery Slot Description 4 displays in timeslot fields on Item Request Detail screen. Without this privilege, this delivery slot description does not display in the timeslot dropdowns.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Edit Item Request	Handheld	Item Requests	With this permission the Edit Item Request menu option on the Item Requests Menu will be displayed. Without this permission the menu option will not be displayed.
Edit Item Request	PC	Item Requests	With this permission, when the user double clicks on an existing Item Request in the Item Request List Screen, the Item Request Detail screen will open with the Item Request and allow the user to make changes. Without this permission, when the user double clicks on an existing Item Request, the Item Request Detail screen will open and the user will only be allowed to view the information and not make any changes.
Request Items	PC	Item Requests	With this permission, the **Request** button on the Item Request Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Request Item Request	Handheld	Item Requests	With this permission the Request Item Request menu option on the Item Requests Menu will be displayed. Without this permission the menu option will not be displayed. With this permission the Request Now menu option on the Item Requests Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Access Item Tickets	Handheld	Item Tickets	With this permission the Item Tickets menu option on the Inv. Management Menu will be displayed. Without this permission the menu option will not be displayed.
Access Item Tickets	PC	Item Tickets	With this permission, the **Item Tickets** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Apply PO to Item Tickets	PC	Item Tickets	With this permission, the **Apply PO** button on the Item Tickets List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Item Tickets	PC	Item Tickets	With this permission, the **Create** button on the Item Tickets List will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delete Item Tickets	PC	Item Tickets	With this permission, the **Delete** button on the Item Tickets List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Item Tickets	PC	Item Tickets	With this permission, when the user double clicks on an existing Item Ticket in the Item Ticket List Screen, the Item Ticket Detail screen will open with the Item Ticket and allow the user to make changes. Without this permission, when the user double clicks on an existing Item Ticket, the Item Ticket Detail screen will open and the user will only be allowed to view the information and not make any changes.
Print Item Tickets	PC	Item Tickets	With this permission, the **Print Tickets** button on the Item Tickets List Screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, the **Print Tickets** button on the Item Tickets Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Update Stock On Hand	PC	Item Tickets	With this permission, the **Update SOH** button on the Item Tickets List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Pick Lists	Handheld	Pick Lists	With this permission the Pick List menu option on the Inv. Management Menu will be displayed. Without this permission the menu option will not be displayed.
Access Pick Lists	PC	Pick Lists	With this permission, the **Pick List** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Action Pick List	Handheld	Pick Lists	With this permission the Action Pick menu option on the Pick List Menu will be displayed. Without this permission the menu option will not be displayed.
Create Pick Lists	Handheld	Pick Lists	With this permission the Within Day Pick and the End Of Day Pick menu options on the Pick List Menu will be displayed. Without this permission the menu options will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Create Pick Lists	PC	Pick Lists	With this permission, the **Create** button on the Pick List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete Pick Lists	PC	Pick Lists	With this permission, the **Delete** button on the Pick List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Price Changes	PC	Price Changes	With this permission, the **Price Change** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Price Changes	PC	Price Changes	With this permission, the **Create** button on the Price Change List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Print Item Tickets For Price Changes	PC	Price Changes	With this permission, the **Item Tickets** button on the Price Change List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Print Shelf Labels For Price Changes	PC	Price Changes	With this permission, the **Shelf Labels** button on the Price Change List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Returns	Handheld	Returns	With this permission the Returns menu option on the Shipping/Receiving Screen will be displayed. Without this permission the menu option will not be displayed.
Access Returns	PC	Returns	With this permission, the **Returns** button on the Shipping/Receiving Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Items To Returns	PC	Returns	With this permission, the Edit Returns permission and with the corresponding data permission for the source of the return, the **Add Items** button on the Return Detail screen will be displayed and enabled when editing an existing Return. With this permission and with the lack of the corresponding data permission for the source of the return or without this permission, the button will not be displayed. This permission does not apply when creating a return. **Add Item** button will always be available when creating.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Add or Edit Items for Return	Handheld	Returns	With this permission and the Edit Returns permission (handheld), the Add/Edit Item menu option on the Return Summary Screen will be displayed. Without this permission the menu option will not be displayed when Editing a Return. This permission does not apply when creating a return. Users will always be able to add and edit items when creating a return.
Create or Edit Return Context Field	PC/Handheld	Returns	Allow the user to create a return with a context field.
Create Returns	Handheld	Returns	With this permission the Create Return menu option on the Returns Menu will be displayed. Without this permission the menu option will not be displayed.
Create Returns	PC	Returns	With this permission, the **Create** button on the Return List Screen will be displayed and enabled. The **Add Item** button will also be enabled on the Return Detail screen. Without this permission, the button will not be displayed.
Create Returns for External Finisher	PC/Handheld	Returns	Allow the user to create a return for an external finisher.
Delete from Returns	PC	Returns	With this permission, (and the corresponding Edit Returns permission) the **Delete** button on the Return Detail screen will be displayed and enabled. Without this permission, the button on the Return Detail screen will not be displayed.
Delete Returns	Handheld	Returns	With this permission the Delete Return menu option on the Returns Menu will be displayed. Without this permission the menu option will not be displayed. See the Returns section for the specifics on how the application will work with the data permissions.
Delete Returns	PC	Returns	With this permission, the **Delete** button on the Return List Screen will be displayed and enabled. Without this permission, the button will not be displayed. If the button is displayed, the user must also have the necessary data permission for the source type on the return that is attempted to be deleted. If the user is not authorized for the source on the return, **User is not authorized to delete this Return.** is displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Dispatch Returns	Handheld	Returns	With this permission the Dispatch Return menu option on the Returns Menu will be displayed. Without this permission the menu option will not be displayed. See the Returns section for the specifics on how the application will work with the data permissions. With this permission the Dispatch Now menu option on the Return Summary Screen will be displayed, when coming from either Create Return or Edit Return. Without this permission the menu option will not be displayed.
Dispatch Returns	PC	Returns	With this permission, the **Dispatch** button on the Return List Screen will be displayed and enabled. Without this permission, the button will not be displayed. If the button is displayed, the user must also have the necessary data permission for the source type on the return that is attempted to be dispatched. If the user is not authorized for the source on the return, **User is not authorized to dispatch this Return.** is displayed. With this permission and the corresponding data permission for the source of the return, the **Dispatch** button on the Return Detail Screen will be displayed and enabled. With this permission and the lack of the corresponding data permission for the source of the return, the **Dispatch** button on the Return Detail Screen will not be displayed. Without this permission, the button will not be displayed.
Edit Return Bill of Lading	PC	Returns	This permission allows the SIM user access to edit a bill of lading associated with a return.
Edit Returns	Handheld	Returns	With this permission the Edit Return menu option on the Returns Menu will be displayed. Without this permission the menu option will not be displayed. See the Returns section for the specifics on how the application will work with the data permissions.
Edit Returns	PC	Returns	With this permission and the corresponding data permission for the source of the return, when the user double-clicks on an existing return in the Return List Screen, the Return Detail screen will open with the Return and allow the user to make changes. With the permission and with the lack of the corresponding data permission for the source of the return or without this permission, when the user double-clicks on an existing Return, the Return Detail screen will open and the user will only be allowed to view the information and not make any changes.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
View Return Bill of Lading	PC	Returns	This permission allows the SIM user access to view a bill of lading associated with a return. Without the view permission, the user cannot edit the bill of lading. The BOL button will not be displayed, but information will be captured.
View Return Details	Handheld	Returns	With this permission the View Details menu option on the Return Summary Screen will be displayed, when coming from either Create Return or Edit Return. Without this permission the menu option will not be displayed.
Access Password Configuration	PC	Security	With this permission, the **Password Configuration** button on the Security Menu will be displayed and enabled. Without this permission, the button will not be displayed.
Access Role Maintenance	PC	Security	With this permission, the **Role Maintenance** button on the Security Menu will be displayed and enabled. Without this permission, the button will not be displayed.
Access Security	PC	Security	With this permission, the **Security** button on the Admin Menu will be displayed and enabled. Without this permission, the button will not be displayed.
Access User Maintenance	PC	Security	With this permission, the **User Maintenance** button on the Security Menu will be displayed and enabled. Without this permission, the button will not be displayed.
Assign Roles to Users	PC	Security	With this permission, the **Assign Roles** button on the Security Menu will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, the **Assign Roles** button on the User Maintenance screen will be displayed and enabled. Without this permission, the button will not be displayed.
Assign Stores to User	PC	Security	With this permission, the **Assign Stores** button on the Security Menu will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, the **Assign Stores** button on the User Maintenance screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delete Role	PC	Security	With this permission, the **Delete** button on the Role List screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, the **Delete** button on the Role Maintenance screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Sequencing	Handheld	Sequencing	With this permission the Sequencing menu option on the Inv. Management Menu will be displayed. Without this permission the menu option will not be displayed.
Access Sequencing	PC	Sequencing	With this permission, the **Sequencing** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Item to Location	PC	Sequencing	With this permission, the **Add** button on the Micro Sequence Edit Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Locations for an Item	PC	Sequencing	With this permission, the **Add Locations** button on the Item Location List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Sequencing Locations	PC	Sequencing	With this permission, the **Add Locations** button on the Macro Sequence Edit screen will be displayed and enabled. Without this permission, the button will not be displayed.
Apply Class List to Location	PC	Sequencing	With this permission, the **Apply Class List** button on the Macro Sequence Edit screen will be displayed and enabled. Without this permission, the button will not be displayed.
Apply Item List to Location	PC	Sequencing	With this permission, the **Apply Item List** button on the Micro Sequence Edit Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Arrange Items Within Location	PC	Sequencing	With this permission, the **Move Up** and **Move Down** buttons on the Micro Sequence Edit Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Arrange Sequencing Locations	PC	Sequencing	With this permission, the **Move Up** and **Move Down** buttons on the Macro Sequence Edit screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete Items from a Location	PC	Sequencing	With this permission, the **Delete** button on the Micro Sequence Edit screen will be displayed and enabled. Without this permission, this button will not be displayed.
Delete Locations for an Item	PC	Sequencing	With this permission, the **Delete** button on the Item Location List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete Sequencing Locations	PC	Sequencing	With this permission, the **Delete** button on the Macro Sequence Edit screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Items Within a Location	PC	Sequencing	With this permission, when the user double clicks on an existing Sequencing Location in the Macro Sequence List Screen, the Micro Sequence List screen will open with the Location and allow the user to make changes. Without this permission, when the user double clicks on an existing Sequencing Location, the Micro Sequence List screen will open and the user will only be allowed to view the information and not make any changes. With this permission, when the user double clicks on an existing Item in the Micro Sequence List Screen the Item Location List screen will open and allow the user to make changes. The **Add Locations** button will also be enabled. Without this permission, when the user double clicks on an existing Item, the Micro Sequence List screen will open and the user will only be allowed to view the information and not make any changes. With this permission, the **Edit Items** button on the Micro Sequence List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Sequencing Locations	PC	Sequencing	With this permission, the **Edit Locations** button on the Macro Sequence List Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Sequence Items	Handheld	Sequencing	With this permission the Sequence Items menu option on the Sequencing Menu will be displayed. Without this permission the menu option will not be displayed.
Sequence Items Within a Location	Handheld	Sequencing	With this permission the Sequence all items in a location menu option on the Sequencing Menu will be displayed. Without this permission the menu option will not be displayed.
Access Ad-Hoc Stock Counts	Handheld	Stock Counts	With this permission the Ad Hoc Stock Count menu option on the Stock Counting Menu will be displayed. Without this permission the menu option will not be displayed.
Access Ad Hoc Stock Counts	PC	Stock Counts	With this permission, the **Ad Hoc Stock Counts** button on the Admin Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Stock Counts	Handheld	Stock Counts	With this permission the Stock Counts menu option on the Inv. Management Menu will be displayed. Without this permission the menu option will not be displayed.
Access Stock Counts	PC	Stock Counts	With this permission, the **Stock Counts** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Amend Stock Count	Handheld	Stock Counts	With this permission the <amend> option for stock counts should be displayed and the Shift-5 function should take the user to the Amend process. Without this permission the <amend> option should not be displayed and the Shift-5 function should not take the user to the Amend process. Nothing should happen when this button combination is clicked.
Amend Stock Re-Count	Handheld	Stock Counts	With this permission the <amend> option for stock re-counts should be displayed and the Shift-5 function should take the user to the Amend process. Without this permission the <amend> option should not be displayed and the Shift-5 function should not take the user to the Amend process. Nothing should happen when this button combination is clicked.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Authorize Stock Count	PC	Stock Counts	If the user has edit permissions for all current stock count types (this will be based on the user having Edit Ad Hoc Stock Counts, Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted), with this permission the **Authorize** button on the Child Stock Count List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Complete Ad-Hoc Count	Handheld	Stock Counts	With this permission the Complete Count menu option on the Stock Counting Menu will be displayed. Without this permission the menu option will not be displayed.
Complete Stock Count	PC	Stock Counts	If the user has edit permissions for the current stock count type (this will be based on the user having Edit Ad Hoc Stock Counts, Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted) with this permission, the **Complete** button on the Child Stock Count List, the Stock Count Detail and the Stock Re-Count Detail Screens will be displayed and enabled. Without this permission, the button will not be displayed.
Confirm Authorization	PC	Stock Counts	If the user has edit permissions for the current stock count type (this will be based on the user having Edit Ad Hoc Stock Counts, Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted) with this permission, the **Confirm Authorization** button on the Child Stock Count List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Confirm Child Stock Count	PC	Stock Counts	If the user has edit permissions for the current stock count type (this will be based on the user having Edit Ad Hoc Stock Counts, Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted) with this permission, the **Confirm Child** button on the Stock Count Authorization Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Count Stock Count	Handheld	Stock Counts	With this permission the Stock Count menu option on the Stock Counting Menu will be displayed. Without this permission the menu option will not be displayed.
Create Ad-Hoc Stock Count	Handheld	Stock Counts	With this permission the Create New Count menu option on the Ad-Hoc Stock Count screen will be displayed. Without this permission the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Delete Stock Count	PC	Stock Counts	With this permission, the **Delete** button on the Stock Count List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Ad Hoc Stock Count	PC	Stock Counts	With this permission, when the user double clicks on an existing Ad Hoc Stock Count in the Stock Count List Screen, the appropriate screen will open and allow the user to make changes, if allowed by the business rules. Without this permission, when the user double clicks on an existing Ad Hoc Stock Count, the appropriate screen will open and the user will only be allowed to view the information and not make any changes.
Edit Unit Stock Counts	PC	Stock Counts	With this permission, when the user double clicks on an existing Unit Stock Count in the Stock Count List Screen, the appropriate screen will open and allow the user to make changes, if allowed by the business rules. Without this permission, when the user double clicks on an existing Unit Stock Count, the appropriate screen will open and the user will only be allowed to view the information and not make any changes.
Edit Unit and Amount Stock Counts	PC	Stock Counts	With this permission, when the user double clicks on an existing Unit and Amount Stock Count in the Stock Count List Screen, the appropriate screen will open and allow the user to make changes, if allowed by the business rules. Without this permission, when the user double clicks on an existing Unit and Amount Stock Count, the appropriate screen will open and the user will only be allowed to view the information and not make any changes.
Re-Count Stock Count	Handheld	Stock Counts	With this permission the Stock Re-Count menu option on the Stock Counting Menu will be displayed. Without this permission the menu option will not be displayed.
Rejected Items	PC	Stock Counts	If the user has edit permissions for all current stock count types (this will be based on the user having Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted), with this permission the **Rejected Items** button on the Child Stock Count List Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Save Child Stock Count	PC	Stock Counts	If the user has edit permissions for the current stock count type (this will be based on the user having Edit Ad Hoc Stock Counts, Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted) with this permission, the **Save Child** button on the Stock Count Authorization Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Take Snapshot	PC	Stock Counts	If the user has edit permissions for the current stock count type (this will be based on the user having Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted) with this permission, the **Take Snapshot** button on the Child Stock Count List Screen, the Stock Count Detail and the Stock Re-Count Detail screens will be displayed and enabled. Without this permission, the button will not be displayed.
Update Authorization Quantity	PC	Stock Counts	If the user has edit permissions for all current stock count types (this will be based on the user having Edit Ad Hoc Stock Counts, Edit Unit Stock Counts or Edit Unit and Amount Stock Count permission granted) on the Stock Count List screen, with this permission the **Update Auth Qty** button on the Child Stock Count List and the Stock Count Authorization Screens will be displayed and enabled. Without this permission, the button will not be displayed. **Note:** When this permission is implemented, the **Display Update Auth Qty** button on the stock count authorize screen system parameter can be removed from the system.
Access Store Orders	PC	Store Orders	With this permission, the **Store Orders** button on the Inventory Management Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Item To Store Order	PC	Store Orders	With this permission and the Edit Store Orders permission, the **Add Item** button on the Store Order Detail Screen will be displayed and enabled when editing an existing Store Order. Without this permission, the button will not be displayed. This permission does not apply when creating a Store Order. **Add Item** button will always be available when creating.
Approve Store Order	PC	Store Orders	With this permission, the **Approve** button on the Store Order Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Cancel Item From Store Order	PC	Store Orders	With this permission, the **Cancel Item** button on the Store Order Detail Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Cancel Store Order	PC	Store Orders	With this permission, the **Cancel Order** button on the Store Orders Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Store Order	PC	Store Orders	With this permission, the **Create Order** button on the Store Orders Screen will be displayed and enabled. The **Add Item** button the Store Order Detail Screen will also be enabled. Without this permission, the button will not be displayed.
Edit Store Orders	PC	Store Orders	With this permission, when the user double clicks on an existing Store Order in the Store Orders Screen, the Store Detail screen will open with the Store Order and allow the user to make changes. Without this permission, when the user double clicks on an existing Store Order, the Store Order Detail screen will open and the user will only be allowed to view the information and not make any changes.
Access Transfer Request	Handheld	Transfers	With this permission the Transfer Request menu option on the Transfer Menu will be displayed. Without this permission the menu option will not be displayed.
Access Transfers	Handheld	Transfers	With this permission the Transfers menu option on the Shipping/Receiving Screen will be displayed. Without this permission the menu option will not be displayed.
Access Transfers	PC	Transfers	With this permission, the **Transfers** button on the Shipping/Receiving Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Add Item to Transfer	PC	Transfers	With this permission and the Edit Transfer permission, the **Add Item** button on the Create Transfer Screen will be displayed and enabled when editing an existing transfer. Without this permission, the button will not be displayed.
Add Item to Transfer Receipt	PC	Transfers	With this permission and the Receive Transfer permission, the **Add Item** button on the Receive Transfer Screen will be displayed and enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Add Item to Transfer Request	PC	Transfers	With this permission and the Edit Transfer Request permission, the **Add Item** button on the Transfer Request Screen will be displayed and enabled when editing an existing Transfer Request. Without this permission, the button will not be displayed when editing an existing Transfer Request. This permission does not apply when creating a new transfer request. The **Add Item** button will always be available when creating a transfer request.
Add or Edit Items on Transfer	Handheld	Transfers	With this permission and the Edit Transfer Permission (handheld), the Add/Edit Item menu option on the Transfer Summary Screen will be displayed. Without this permission the menu option will not be displayed when editing transfer. This permission does not apply when creating a transfer. Users will always be able to add and edit items when creating a transfer.
Amend Transfer Receipt	Handheld	Transfers	With this permission the Amend Transfer menu option on the Transfer Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Authorize Transfer Requests	PC	Transfers	With this permission, the **Accept** and **Reject** buttons on the Transfer Response Screen will be displayed and enabled. Without this permission, the buttons will not be displayed.
Complete Transfer Receipt	Handheld	Transfers	With this permission the Complete Transfer menu option on the Transfer Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Confirm Transfer	PC	Transfers	With this permission, the **Confirm** button on the Receive Transfer Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Create Transfer Requests	Handheld	Transfers	With this permission the Create Tsf Request menu option on the Transfer Request Screen will be displayed. Without this permission the menu option will not be displayed.
Create Transfer Requests	PC	Transfers	With this permission, the **Create Request** button on the Transfer List Screen will be displayed and enabled. The **Add Item** button on the Transfer Request Detail screen will also be enabled. Without this permission, the button will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Create Transfers	Handheld	Transfers	With this permission the Create Transfer menu option on the Transfer Menu will be displayed. Without this permission the menu option will not be displayed.
Create Transfers	PC	Transfers	With this permission, the **Create** button on the Transfer List Screen will be displayed and enabled. The **Add Item** button on the Create Transfer Screen will also be enabled. Without this permission, the button will not be displayed.
Delete From Transfer	PC	Transfers	With this permission, the **Delete** button on the Create Transfer Screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission, the **Delete** button on the Receive Transfer Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete From Transfer Request	PC	Transfers	With this permission, the **Delete** button on the Transfer Request Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete Items from Transfer	Handheld	Transfers	With this permission the Delete Item menu option on the Transfer Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Delete Transfer Request	Handheld	Transfers	With this permission the Delete Tsf Request menu option on the Transfer Request Screen will be displayed. Without this permission the menu option will not be displayed.
Delete Transfers	Handheld	Transfers	With this permission the Delete Transfer menu option on the Transfer Menu will be displayed. Without this permission the menu option will not be displayed. With this permission the Delete Transfer menu option on the Delete Transfer Screen will be displayed. Without this permission the menu option will not be displayed.
Delete Transfers	PC	Transfers	With this permission, the **Delete** button on the Transfer List Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Transfer Bill of Lading	PC	Transfer	This permission allows the SIM user access to edit a bill of lading associated with a transfer.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Edit Transfers	Handheld	Transfers	With this permission the Edit Transfer menu option on the Transfer Menu will be displayed. Without this permission the menu option will not be displayed.
Edit Transfers	PC	Transfers	With this permission, when the user double-clicks on an existing Transfer in the Transfer List Screen, the user will be allowed to make changes. Without this permission, when the user double-clicks on an existing Transfer, the appropriate screen will open and the user will only be allowed to view the information and not make any changes.
Edit Transfer Requests	Handheld	Transfers	With this permission the Edit Tsf Request menu option on the Transfer Request Screen will be displayed. Without this permission the menu option will not be displayed.
Edit Transfer Requests	PC	Transfers	With this permission, when the user double-clicks on an existing Transfer Request in the Transfer List Screen, the user will be allowed to make changes. Without this permission, when the user double clicks on an existing Transfer Request, the appropriate screen will open and the user will only be allowed to view the information and not make any changes.
Receive All Items on Transfer	Handheld	Transfers	With this permission the Receive All menu option on the Transfer In Screen will be displayed. Without this permission the menu option will not be displayed.
Receive All on Transfer	PC	Transfers	With this permission, the **Receive All** button on the Receive Transfer Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Receive Item on Transfer	Handheld	Transfers	With this permission the Receive Item menu option on the Transfer In Screen will be displayed. Without this permission the menu option will not be displayed.
Receive Transfers	Handheld	Transfers	With this permission the Receive Transfer menu option on the Transfer Menu will be displayed. Without this permission the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Receive Transfers	PC	Transfers	With this permission, when the user double-clicks on an existing Dispatched Transfer in the Transfer List Screen, the Receive Transfer screen will open with the Transfer and allow the user to make changes. Without this permission, when the user double-clicks on an existing Dispatched Transfer, the Receive Transfer screen will open and the user will only be allowed to view the information and not make any changes.
Record Transfer Damages	Handheld	Transfers	With this permission the Record Damages menu option on the Transfer Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Request Transfer Request	Handheld	Transfers	With this permission the Request Tsf Request menu option on the Transfer Request Screen will be displayed. Without this permission the menu option will not be displayed.
Request Transfer Request	PC	Transfers	With this permission, the **Request** button on the Transfer Request Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Transfer Context Field	PC	Transfers	With this permission, the user can add and modify a context field (status can be New Request or In Progress). Without this permission, the workflow will be bypassed and the user cannot modify the context field. The user also cannot access the context field from the main transfer request field.
Transfer Context Field	HH	Transfers	With this permission, the user can add and modify a context field (status can be New Request or In Progress). Without this permission, the workflow will be bypassed and the user cannot modify the context field. The user also cannot access the context field from the main transfer request field.
View Details for Transfer	Handheld	Transfers	With this permission the View Details menu option on the Transfer Summary Screen will be displayed, when coming from either Create Transfer or Edit Transfer. Without this permission the menu option will not be displayed. With this permission the View Details menu option on the Delete Transfer Screen will be displayed. Without this permission the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
View Transfer Bill of Lading	PC	Transfer	This permission allows the SIM user access to view a bill of lading associated with a transfer. Without the view permission, the user cannot edit the bill of lading. The BOL button will not be displayed, but information will be captured.
Access UIN Attributes	PC	UIN	With this permission, the **UIN Attributes** button on the Setup Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access UIN Resolution	PC	UIN	With this permission the **UIN Resolution** button on the Admin screen will be displayed and enabled. Without this permission, the button will not be displayed.
Print AGSN Ticket	Handheld	UIN	With this privilege, the user is able to print AGSNs from the Item Ticket screen and UIN Detail screen. Without this privilege, the Print option will not be displayed.
Print AGSN Ticket	PC	UIN	With this privilege, the user is able to print AGSNs from the Item Ticket screen and UIN Detail screen.
Resolve UIN Exceptions	PC	UIN	With this permission the **Resolve** button on the UIN Resolution List screen will be displayed and enabled. Without this permission, the button will not be displayed.
Update UIN Status	PC	UIN	With this permission the **Update Status** button on the UIN Resolution List Screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission the **Update Status** button on the UIN Detail Screen (accessed from either item lookup) will be displayed and enabled. Without this permission, the button will not be displayed.
View UIN Detail	PC	UIN	With this permission the **UIN Detail** button on the Item Lookup screen will be displayed and enabled. Without this permission, the button will not be displayed. With this permission the UIN Detail tab on the Item Lookup popup search screen will be displayed and enabled. Without this permission the tab will be disabled.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
View UIN History	PC	UIN	With this privilege the View History button on the UIN Resolution List Screen is displayed and enabled. Without this privilege the button will not be displayed. With this privilege the View History button on the UIN Detail Screen (accessed from item lookup) is displayed and enabled. Without this privilege the button will not be displayed.
Access Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Warehouse Delivery menu option on the Shipping/Receiving Screen will be displayed. Without this permission the menu option will not be displayed.
Access Warehouse Delivery	PC	Warehouse Delivery	With this permission, the **Warehouse Delivery** button on the Shipping/Receiving Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Access Warehouse Quick Receiving	Handheld	Warehouse Delivery	With this permission the Warehouse Quick Receiving menu option on the Shipping/Receiving Screen will be displayed. Without this permission the menu option will not be displayed.
Add Item to Warehouse Delivery	PC	Warehouse Delivery	With this permission, the **Add Item** button on the Receive Case Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Amend Container	Handheld	Warehouse Delivery	With this permission the Amend menu option on the Identify Container Screen will be displayed. Without this permission the menu option will not be displayed.
Amend Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Amend Delivery menu option on the Delivery Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Cancel Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Cancel Delivery menu option on the Warehouse Delivery Screen will be displayed. Without this permission the menu option will not be displayed.
Confirm Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Confirm Delivery menu option on the Delivery Summary Screen will be displayed. Without this permission the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Confirm Warehouse Delivery	PC	Warehouse Delivery	With this permission, the **Confirm** button on the Receive Container Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Delete from Warehouse Delivery	PC	Warehouse Delivery	With this permission, the **Delete** button on the Receive Case Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Edit Warehouse Delivery	PC	Warehouse Delivery	With this permission, when the user double-clicks on an existing Warehouse Delivery in the Warehouse Delivery List Screen, the Warehouse Delivery Detail screen will open with the Warehouse Delivery and allow the user to make changes. Without this permission, when the user double-clicks on an existing Warehouse Delivery, the Warehouse Delivery Detail screen will open and the user will only be allowed to view the information and not make any changes.
Receive All Containers for Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Receive All menu option on the Delivery Information Screen will be displayed. Without this permission the menu option will not be displayed.
Receive Case Level for Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Receive Container menu option on the Container Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Receive Item Level for Warehouse Delivery	Handheld	Warehouse Delivery	With this permission the Receive Case Level menu option on the Container Summary Screen will be displayed. Without this permission the menu option will not be displayed.
Receive Warehouse Delivery	PC	Warehouse Delivery	With this permission, the **Receive** button on the Receive Container Screen will be displayed and enabled. Without this permission, the button will not be displayed.
Record Damages	Handheld	Warehouse Delivery	With this permission the Record Damages menu option on the Container Summary Screen will be displayed. Without this permission the menu option will not be displayed.

Table A–1 (Cont.) SIM Permissions

Permission	Type	Topic	Usage
Record Missing Container	Handheld	Warehouse Delivery	With this permission the Record Missing menu option on the Discrepancy Alert Screen will be displayed. Without this permission the menu option will not be displayed.
Un-Receive Container	Handheld	Warehouse Delivery	With this permission the Un-Receive menu option on the Identify Container Screen will be displayed. Without this permission the menu option will not be displayed.
Un-Receive Warehouse Delivery	PC	Warehouse Delivery	With this permission, the **Un-Receive** button on the Receive Container Screen will be displayed and enabled. Without this permission, the button will not be displayed.

B

Appendix: LDAP Schema

This appendix discusses the object classes specified for the SIM application security model. The LDIF file used to create the object classes can be found in sim_objectclasses.ldif.

For more information, see "Setting up LDAP Data for SIM".

Object Classes

There are four SIM-defined Object Classes:

- simRole
- simStore
- simUser
- simUserRole

They are described as follows:

Table B–1 simRole Object Class

Attribute Name	Mandatory	Description
roleName	Yes	Role Name. Syntax: String.
Type	No	Type of a Role – Store or Corporate. Syntax: String.
Description	No	Description of a Role. Syntax: String.

Table B–2 simStore Object Class

Attribute Name	Mandatory	Description
storeID	Yes	Store ID. Syntax: String.

Table B–3 simUser Object Class

Attribute Name	Mandatory	Description
superUser	Yes	Is user a superuser? Syntax: Boolean (TRUE or FALSE)
empStatus	Yes	Employee's status (0 = active, 1 = inactive, 2 = deleted, 3 = locked) Syntax: Integer
preferredCountry	No	Preferred country code Syntax: String
preferredLanguage	No	Preferred language code Syntax: String
Mail	No	E-mail address. Syntax: String
telephoneNumber	No	Telephone number. Syntax: String
externalID	No	External system ID. Syntax: String
Supervisor	No	Supervisor Syntax: String
description	No	Descriptions or comments. Syntax: String
startTimestamp	No	Start date Syntax: Generalized Time
endTimestamp	No	End date Syntax: Generalized Time
defaultStore	No	DN of the default store Syntax: String
userStores	No	DN of User's stores, multiple values. Syntax: String This attribute is only used if the user is not a super-user. Super-users do not use store assignments.

Table B–4 simUserRole Object Class

Attribute Name	Mandatory	Description
roleName	Yes	Role assignment name. Syntax: String
userRole	Yes	DN of role Syntax: String

Table B–4 (Cont.) simUserRole Object Class

Attribute Name	Mandatory	Description
userRoleStores	Yes	DN of stores that user role is assigned, multiple values. Syntax: String
StartTimestamp	No	Start time Syntax: Generalized Time
EndTimestamp	No	End time Syntax: Generalized Time

Directory Entry Structure

For this example, the name of the retail company is **MyCompany** and the parent directory of the SIM entries is **cn=SIM,dc=mycompany,dc=com**.

There are two subtrees for Roles and Stores:

```
cn=SIMRoles,cn=SIM,dc=mycompany,dc=com
cn=SIMStores,cn=SIM,dc=mycompany,dc=com
```

Users are stored in the following directory:

```
cn=Users,dc=mycompany,dc=com
```

Configuration File ldap.cfg

A configuration file called ladp.cfg is located in the SIM Server at sim-home/files/prod/config. See Chapter 2, "Backend System Configuration" in *Oracle Retail Store Inventory Management Operations Guide* for more information.

The keys SIM_DN and BASE_DN are defined in ldap.cfg. The BASE_DN is the directory where the User container is located; and the SIM_DN directory contains the parent directories for the Role and Store. For example:

```
BASE_DN= dc=mycompany,dc=com
SIM_DN= cn=SIM,dc=mycompany,dc=com
```

Sample LDIF Data Files

Sample data entries are described in this section.

For this example, the name of the retail company is **MyCompany** and the parent directory of the SIM entries is **cn=SIM,dc=mycompany,dc=com**.

Store

DN of Store:

```
storeId=xxxx,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
```

Where *xxxx* is a store ID. The following is a sample LDIF file that adds the entry for Store 7000:

```
dn: storeId=7000,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
changetype: addobject
Class: simStore
storeId: 7000
```

Role

DN of Role:

```
roleName=xxxx,cn=SIMRoles,cn=SIM,dc=mycompany,dc=com
```

Where *xxxx* is a roleName defined in the SIM database (ac_role.name). The following is a sample LDIF file that adds the entries for ADMINISTRATOR and MANAGER:

```
dn: roleName=ADMINISTRATOR,cn=SIMRoles,cn=SIM,dc=mycompany,dc=com
changetype: addobject
Class: simRole
roleName: ADMINISTRATOR
type: Corporate
description: Corporate Administrator

dn: roleName=MANAGER,cn=SIMRoles,cn=SIM,dc=mycompany,dc=com
changetype: addobject
Class: simRole
roleName: MANAGER
type: Store
description: Store Manager
```

User

DN of User:

```
cn=xxxx,cn=Users,dc=mycompany,dc=com
```

Where *xxxx* is the username of an user. The following is a sample LDIF file that adds a User Entry. The username is **superuser1** and the default store is **7000**, and has access to stores 7000, 7010 and 7011.

> **Note:** The attributes **cn** and **uid** should be the same, and are the login ID of the user.

```
dn: cn=superuser1,cn=Users,dc=mycompany,dc=com
changetype: addobject
class: top
objectclass: organizationalpers
onobjectclass: orcluser
objectclass: person
objectclass: orcluserv2
objectclass: inetorgperson
objectclass: simUser
cn: superuser1
uid: superuser1
superUser: TRUE
empStatus: 0
preferredCountry: US
preferredLanguage: en
givenname: superuser1
middleName: M1
sn: Superuser1
mail: superuser1@mycompany.com
telephoneNumber: 800-111-2222
externalId: superuser1
supervisor: X
description: SIM Store ID 7000 Super User.
```

```
startTimestamp: 20071026000000Z#
endTimestamp:
defaultStore: storeId=7000,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
userStores: storeId=7000,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
userStores: storeId=7010,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
userStores: storeId=7011,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
userpassword: welcome1
```

User's Role

DN of user role:

```
roleNa
me=xxxx,cn=user1,cn=Users,dc=mycompany,dc=com
```

Where xxxx is the role assigned to an user with username user1. The following is a sample LDIF file that will add an entry for MANAGER role for user User1:

```
dn: roleName=MANAGER,cn=user1,cn=Users,dc=mycompany,dc=com
changetype: add
objectclass: simUserRole
roleName: MANAGER
userRole: roleName=MANAGER,cn=SIMRoles,cn=SIM,dc=mycompany,dc=com
userRoleStores: storeId=7000,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
userRoleStores: storeId=7010,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
userRoleStores: storeId=7011,cn=SIMStores,cn=SIM,dc=mycompany,dc=com
```

C

Appendix: Realtime Oracle Retail Point-of-Service Updates

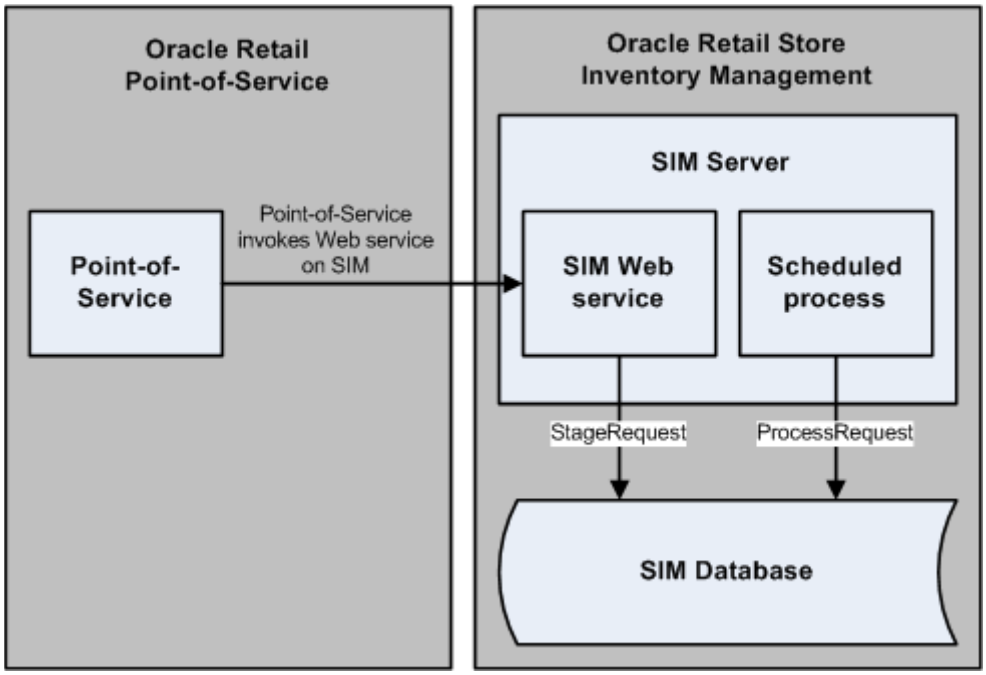
SIM is updated with Oracle Retail Point-of-Service transaction information on a periodic basis.

Near-real-time updates in SIM will enable the following:

- A near-real-time interface allows Oracle Retail Point-of-Service to transfer in transaction information and update inventory.
- SIM requires an audit trail to understand the process that updated the sale, and a purging process for such an audit trail must be implemented.
- Update features for snapshots, physical count quantities and authorized values.
- UOM conversion, to convert from selling UOM to standard UOM.

With near-real-time updates, SIM inventory will be up-to-date with Oracle Retail Point-of-Service inventory. Every transaction that takes place at Point-of-Service is posted to SIM using a web service. The whole process is a near-real-time update. The call from Point-of-Service to SIM using a web service will be a blocking call. Therefore, the web service performs minimal processing and persists the transaction data to staging tables.

Figure C–1 Real-time Updates Process Flow



Configuration

When the real time updates path from Oracle Retail Point-of-Service is enabled, the config parameter STOCK_COUNT_SALES_PROCESSING is always set to Timestamp processing to account for late sales and open stock counts.

The polling timer for polling the tables is configurable. A new entry needs to be added in POLLING_TIMER table:

```
INSERT INTO POLLING_TIMER (ID, MESSAGE_DIRECTION, MESSAGE_FAMILY, ENABLED) VALUES
(POLLING_TIMER_SEQ.nextval, 'Inbound', 'SALERETTXN', 'Y');
```

A system parameter (days to hold sales posting record) is added, which will be used while purging records:

```
INSERT INTO RK_CONFIG (CONFIG_KEY, CONFIG_VALUE, CONFIG_TYPE, TOPIC_KEY, IS_
EDITABLE) values ('DAYS_TO_HOLD_SALES_POSTING',
24,'java.lang.Integer','PURGE','N');
```

Audit/Logging

All transactions posted from Oracle Retail Point-of-Service first get staged to the STAGED_MESSAGE table.

The polling timer framework picks up records from the STAGED_MESSAGE and posts it to the POS_TRANSACTION table. This table acts as the audit for Oracle Retail Point-of-Service transactions. Any sale/return/void sale/void return gets posted to this table. and if any audit needs to be performed, this is the table which consists of the records.

The POS_TRANSACTION table has a record for every item in the transaction. It is a denormalized view of the transaction posted to SIM.

POS_TRANSACTION has three flags:

- SALE_RET_TXN_PROCESS_SUCCESS – This indicates whether the sale/return inventory update processing was successful or not.
- INV_RESV_PROCESS_SUCCESS – This indicates if the inventory reservation process was successful or not.
- UIN_PROCESS_SUCCESS – This indicates if the UIN processing was successful or not.

The previous three flags are needed because it is possible that Oracle Retail Point-of-Service that invokes the web service does not have inventory reservation turned on, or it is possible that only UIN processing is enabled, but sale/return transactions are not posted. The transaction request sent by Point-of-Service will have parameters in the web service request which helps SIM identify which of the processes are turned on or off.

If SALE_RETURN_TXN_PROCESS_SUCCESS is **null**, that means the Sale return processing is not turned on by Point-of-Service, and therefore inventory updates in SIM for Sale/return/void sale and void return actions should not take place. However, if the value is **N**, that means that processing should have happened and could not take place successfully due to an error. The error would be present in the ERROR_MSG column of the same table.

Return/Void Item Disposition

Near-real-time transaction updates in SIM support the item disposition for return/void transactions based on the reason code mapping in SIM.

Point-of-Service maps the SIM reason codes, and the reason code is sent to SIM in the web service message for the return and post void transactions. These are SIM inventory adjustment reason codes mapped to the item disposition. Based on the disposition mapped to the reason code, SIM moves the inventory buckets accordingly.

For Void of Sale, if the web service returns the reason code, and the Enable Item Disposition in the Transaction Updates store parameter is set to Yes, then the system updates the inventory bucket based on the disposition mapped to that reason code. If there is no reason code sent, the SOH is incremented. This is done with an inventory adjustment:

- +Unavailable – This moves the stock from ATS (Available) to TRBL (Unavailable).
- –SOH – This moves the stock from ATS (Available) to DIST (Out).

If the reason code received is invalid/not present/mapped incorrectly, the system writes an error log and continues to process the transaction.

Note: Point-of-Service must capture and publish the reason codes for the return/void transactions to SIM in the web service call.

UIN Processing

If the transaction contains a UIN, the status of the UIN is first updated to In Stock and an inventory adjustment is created based on the reason code sent.

If the Item Disposition is +Unavailable, the status of the UIN is changed to Unavailable.

If the Item Disposition is –SOH, the status of the UIN is changed to Removed from Inventory.

Note: UIN must be in a valid status for the return/void of sale.

D

Appendix: Transfer Localization

Transfers allow a retailer to send inventory from one location to another. Transfer requests provide stores the ability to ask for products from other stores or allow corporate users to move inventory across stores using RMS. SIM allows stores to add, edit, delete or send a request to another store on the PC and Handheld.

Users will only be allowed to accept or reject a transfer request awaiting response on the PC.

Transfer localization offers the following:

- Ability to differentiate stock into different buckets depending on stock status (for example, concept of in-transit stock, reserved for transfer).
- Ability to have a system that automatically updates stock inventory on the basis of the status of the transfer.
- Ability for the sending store to save the transfer and then later return to the transfer and cancel, edit or dispatch the transfer.
- System can be used to transfer stock between stores on the same site (for example, Main Store to PFS). This would include the ability to auto-accept stock without scanning items at the moment the transfer is received.
- Ability to differentiate stock into different buckets depending on stock status (for example, Unavailable stock, Stock in transit).
- Ability to have a system that automatically updates stock inventory based on the status of the transfer.
- Ability to alert when the transfer has not been received within a specific time constraint (report or alert).
- Automatically create corrective transfers of stock to rectify the scenarios where the physically shipped stock does not match the stock recorded on the initial transfer.

Process Requirements

Transfer Zones

SIM enforces transfer zones. Only stores within the same transfer zone can send inventory to each other or create requests for each other.

If a store has a transfer zone of NULL, then SIM allows any store to request from or ship to such a store. That means a NULL transfer zone is a universal store.

This information is populated by RMS.

Auto Receiving

SIM allows certain stores to be setup for auto receiving.

Buddy Stores

SIM allows a partial group of stores to be selected that are preferential entities to ship to from a warehouse.

Transfer Force Close Indicator

This indicator is used only for store-to-store transfers.

- System Admin: Transfer Force Close Indicator for Short Receiving.
  - NL – No Loss
  - SL – Sending Loss
  - RL – Receiving Loss

RMS needs to set up their system to match what SIM has for this system admin setting.

> **Note:** In SIM, the SL and RL attributes appear to function the same. However, once they reach RMS they operate differently.

No Loss

Sending store is incremented or decremented by the overage/shortage.

Table D–1 No Loss Shortage: Shortage is added back to the sending store.

	Sending Store	Receiving Store
Beginning SOH	1000 SOH	1000 SOH, 0 In Transit
Sent qty 50	950 SOH	1000 SOH, 50 In Transit
Received qty 30 (shortage)	970 SOH	1030 SOH, 0 In Transit

Table D–2 Overage: Overage is always deducted from the sending store.

	Sending Store	Receiving Store
Beginning SOH	1000 SOH	1000 SOH, 0 In Transit
Sent qty 50	950 SOH	1000 SOH, 50 In Transit
Received qty 70 (overage)	930 SOH	1070 SOH, 0 In Transit

Sending Loss

For shortages, no perpetual inventory is sent back to the sending store. Sending store is financially responsible.

> **Note:** This is handled on the RMS side.

Table D–3 Sending Loss Shortage: Shortage is not added back.

	Sending Store	Receiving Store
Beginning SOH	1000 SOH	1000 SOH, 0 In Transit
Sent qty 50	950 SOH	1000 SOH, 50 In Transit
Received qty 30 (shortage)	950 SOH	1030 SOH, 0 In Transit

Table D–4 Sending Loss Overage: Overage is deducted from the sending store.

	Sending Store	Receiving Store
Beginning SOH	1000 SOH	1000 SOH, 0 In Transit
Sent qty 50	950 SOH	1000 SOH, 50 In Transit
Received qty 70 (overage)	930 SOH	1070 SOH, 0 In Transit

Receiving Loss

For shortages, no perpetual inventory is sent back to the sending store. Receiving store is financially responsible.

Note: This is handled on the RMS side.

Table D–5 Receiving Loss Shortage: Shortage is not added back.

	Sending Store	Receiving Store
Beginning SOH	1000 SOH	1000 SOH, 0 In Transit
Sent qty 50	950 SOH	1000 SOH, 50 In Transit
Received qty 30 (shortage)	950 SOH	1030 SOH, 0 In Transit

Table D–6 Overage: Overage is deducted from the sending store.

	Sending Store	Receiving Store
Beginning SOH	1000 SOH	1000 SOH, 0 In Transit
Sent qty 50	950 SOH	1000 SOH, 50 In Transit
Received qty 70 (overage)	930 SOH	1070 SOH, 0 In Transit

Receive Entire Transfer Parameter

When this parameter is set to **No**, then the Transfer detail dialog allows the user to receive more or less. If the parameter is set to **Yes**, then the dialog is modified to only allow the user to receive the entire transfer.

Store Receiving

The receiving store in SIM will see the shipped quantities immediately on dispatch by the sending store. This process will update the in transit quantities and reduce outstanding transfer values.

Note: SIM can be configured to not allow some users to start receiving these transactions, but this is handled through security.

Dispatching a Transfer

The transfer reserved quantity for the outbound location will decrease as will the stock on hand for the outbound location.

The receiving location will have its in-transit bucket updated with the ship quantity. The transfer quantity is removed from the in-transit bucket to the stock-on-hand bucket when the receiving store receives the transfer.

The transaction must be recorded in the staging table. A record is written for the sending store when the transfer is dispatched and a record is written when the transfer is received by the receiving store.

If the transfer contains packs then the stock movement should follow additional rules:

- Non-sellable simple packs – adjust the quantity at the component level. The pack number will not hold inventory. For example: If a transfer contains Pack A with a transfer quantity of two, and Pack A is made up of Item B (quantity of three), then when the transfer is dispatched, the stock-on-hand for Item B at the sending store is decremented by six (two units of Pack A multiplied by three units of Item B). The in-transit bucket at the receiving store, for Item B, is incremented by six.
- Non-sellable complex packs - adjust the quantity at the component level. The pack number will not hold inventory. For example: If a transfer contains Pack A with a transfer quantity of two, and Pack A is made up of Item B and Item C, then when the transfer is dispatched, the stock-on-hand for Item B at the sending store is decremented by two (two units of Pack A multiplied by one unit of Item B), and Item C is decremented by two. The in-transit bucket at the receiving store is incremented for each item by the respective amount.
- Sellable simple packs – adjust the quantity at the pack level. For example: If Pack A is a sellable pack and a transfer is dispatched with a transfer quantity of five, then the stock-on-hand at the sending store for Pack A is decremented by five, and the in-transit quantity for the receiving store is incremented by five.
- Sellable complex pack – adjust the quantity at the pack level. Has the same stock movement as the sellable simple pack.
- If the transfer's receiving location is an auto close location, then the transfer is considered received when it is dispatched from the sending store:
  - The status of the transfer is **Received**.
  - The received quantity is equal to the transfer quantity.
  - The delivery date and close date equal the dispatch date.
  - A record for the receiving store is inserted into the staging table.
  - Stock-on-hand will be incremented at the receiving location and decremented at the sending location. The in-transit bucket is not affected.

E

Appendix: UPC Barcode

UPC-E items compress a normal 12-digit UPC-A item into six digits. SIM has the ability to decompress UPC-E barcodes to UPC-A. A seventh digit acts as a check digit for the UPC-E number. When the user scans the UPC-E barcode, SIM finds the UPC-A barcode and displays the item ID associated with it.

Differences Between UPC-A and UPC-E

UPC-E is also called zero suppressed UPC because UPC-E compresses a normal twelve-digit UPC-A number into a six-digit code by suppressing the number system digit, trailing zeros in the manufacturers code and leading zeros in the product identification part of the bar code message. A seventh check digit is encoded into a parity pattern for the six main digits. UPC-E can thus be uncompressed back into a standard UPC-A twelve-digit number.

> **Note:** Most bar code readers can be configured to automatically convert six-digit UPC-E numbers to twelve-digit UPC-A numbers before they are transmitted to a host computer.

The main difference between a UPC-A symbol and a UPC-E symbol is the size. The following image presents a UPC-A bar code (left) and the same data encoded as a UPC-E bar code (right):

Figure E–1 UPC-A and UPC-E Differences

To convert between UPC-A and UPC-E bar code numbers, you can use the following table or try online UPC-E converter program. In the following, the number **0** and each of the letters (**a**, **b**, **c**, **d** and **e**) represent individual digits in the bar code message. The letter **X** represents the UPC check digit.

Table E–1 UPC Conversion Table

UPC-A Number	Equivalent UPC-E	Notes
0ab00000cdeX	abcde0X	Manufacturer code must have two leading digits with three trailing zeros and the item number is limited to three digits (000 to 999).
0ab10000cdeX	abcde1X	Manufacturer code must have three leading digits ending with **1** and two trailing zeros. The item number is limited to three digits.
0ab20000cdeX	abcde2X	Manufacturer code must have three leading digits ending with **2** and two trailing zeros. The item number is limited to three digits.
0abc00000deX	abcde3X	Manufacturer code must have three leading digits and two trailing zeros. The item number is limited to two digits (00 to 99).
0abcd00000eX	abcde4X	Manufacturer code must have four leading digits with one trailing zero and the item number is limited to one digit (0 to 9).
0abcde00005X	abcde5X	Manufacturer code has all five digits. The item number is limited to a single digit consisting of either **5**, **6**, **7**, **8** or **9**.
0abcde00006X	abcde6X	
0abcde00007X	abcde7X	
0abcde00008X	abcde8X	
0abcde00009X	abcde9X	

Conversion Between UPC-A and UPC-E

Not all UPC-A numbers can be compressed to UPC-E. These codes with a corresponding UPC-E code must have at least four zeros. The requirements are:

1. If the manufacturer code ends with 000, 100, or 200, the UPC-E code consists of the first two characters of the manufacturer code, the last three characters of the product code, followed by the third character of the manufacturer code. In this case, the product code must be 00000 and 00999.
2. If the manufacturer code ends with 00 but does not meet the first requirement, the UPC-E code consists of the first three characters of the manufacturer code, the last two characters of the product code, followed by digit **3**. The product code can only contain two digits (00000 to 00099).
3. If the manufacturer code ends in **0** but none of the previous qualifies, the UPC-E consists of the first four digits of the manufacturer code and the last digit of the product code, followed by the digit **4**. The product code in this case can only contain one digit (00000 to 00009).
4. If the manufacturer code ends with non-zero digit, the UPC-E code consists of the manufacturer code and the last digit of the product code. In this case the product case can only be one from 00005 to 00009 because 0 through 4 has been used for the previous four cases.

For a free web-based utility for converting between UPC-A and UPC-E, go to the following URL:

http://www.morovia.com/education/utility/upc-ean.asp

Quick Response Codes

Quick Response (QR) Codes are two dimensional bar codes that can be generated to represent any text, most often a custom URL. They can be read by a number of mobile applications through a phone's camera, and provide a way to hyperlink to the physical world. QR Codes can be placed next to products to provide further information or digital coupons, even if the QR Code is on a billboard or store signage. QR Codes and the associated web sites are controlled by the retailer, thus allowing them to recapture control of product research in their store, as well as provide value added information and digital coupons.

QR code functionality in SIM allows the user to:

- Track the image reference to QR Codes at the item/location/image type/time level
- Generate tickets/labels by location when QR codes change based on time/entry using the existing ticket/label dialogues
- Configure the generation of tickets and/or labels automatically
- Print tickets and labels with QR codes
- Allow external systems to integrate QR references within SIM

By running the batch, SIM creates a ticket or label entry in the ticketing screen following the standard pattern of ticketing and labeling. This is configurable.

F

Appendix: Code Examples

This appendix includes code samples referenced in Example Code in Chapter 7, "Customization".

Example F–1 CustomSerialNumber

```
package oracle.retail.sim.closed.customuin;

import oracle.retail.sim.closed.business.BusinessObject;
import oracle.retail.sim.closed.common.BusinessException;
import oracle.retail.sim.closed.uin.UINStatus;
import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;

/*
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
public class CustomSerialNumber extends BusinessObject {
    private static final long serialVersionUID = 732852159610881681 4L;

    private Long id;
    private UINStatus status;

    public void setId(Long id) {
        if (id != null) {
            this.id = id;
        }
    }

    public Long getId() {
        return id;
    }

    public UINStatus getStatus() {
        return status;
    }

    public void setStatus(UINStatus status) throws BusinessException {
        checkForNullParameter("Status", status);
        executeRule("setStatus", status);
        this.status = status;
    }

    public boolean equals(Object object) {
        if (object == this) {
            return true;
```

```
        }
        if (object == null || object.getClass() != getClass()) {
            return false;
        }
        CustomSerialNumber other = (CustomSerialNumber) object;
        EqualsBuilder builder = new EqualsBuilder();
        builder.append(id, other.id);
        return builder.isEquals();
    }

    public int hashCode() {
        HashCodeBuilder builder = new HashCodeBuilder();
        builder.append(id);
        return builder.toHashCode();
    }
}
```

Example F–2 CustomUINBean

```
package oracle.retail.sim.closed.ejb;

import javax.ejb.*;
import oracle.retail.sim.closed.abstractbean.AbstractSimServiceBean;
import oracle.retail.sim.closed.business.ServerServiceFactory;
import oracle.retail.sim.closed.logging.LogNames;
import oracle.retail.sim.closed.logging.LogService;
import oracle.retail.sim.closed.util.CompressedObject;
import oracle.retail.sim.closed.util.SimSession;
import oracle.retail.sim.closed.util.UniversalContext;

/**
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
@Stateless(mappedName = "CustomUINBean")
public class CustomUINBean extends AbstractSimServiceBean implements
CustomUINInterface {
    public CompressedObject moveSerialNumbersToInStock(CompressedObject
compressed0,
                                CompressedObject compressedSimSession) throws
Exception {
        long startTime = System.currentTimeMillis();
        if (LogService.isDebugEnabled(LogNames.SERVICE_TIMINGS)) {
            LogService.debug(LogNames.SERVICE_TIMINGS,
                  "Starting service method:
CustomUINBean.moveSerialNumbersToInStock()");
        }
        try {
            java.util.List<oracle.retail.sim.closed.custom.CustomSerialNumber>
arg0 =
               (java.util.List<oracle.retail.sim.closed.custom.CustomSerialNumber>)
                      compressed0.recoverObject();
            UniversalContext.setSession((SimSession)
compressedSimSession.recoverObject());
            new CustomerUINServerServices().moveSerialNumbersToInStock(arg0);
            return new CompressedObject(null);
        } catch (Throwable t) {
            handleException(t, true);
            return null;
        } finally {
            completeServiceContext();
```

```
            long endTime = System.currentTimeMillis();
            if (LogService.isDebugEnabled(LogNames.SERVICE_TIMINGS)) {
                LogService.debug(LogNames.SERVICE_TIMINGS, "Took " + (endTime -
startTime)
                    + "ms. for invocation of:
CustomUINBean.moveSerialNumbersToInStock()");
            }
        }
    }

}
```

Example F–3 CustomUINCountTableEditor

```
package oracle.retail.sim.shared.swing.custom;

import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import javax.swing.JComponent;
import javax.swing.JTextField;
import oracle.retail.sim.closed.swing.table.SimPopupTableEditor;
import oracle.retail.sim.closed.swing.table.SimTableEditorEventAdaptor;
import oracle.retail.sim.closed.swing.table.SimTableEditorListener;
import oracle.retail.sim.closed.swing.tableeditor.PopupTableEditorListener;

/*******************************************************************************
***********
 * Table editor that handles a boolean true/false value, displays a non-editable
check box.
 * If the cell
 * is clicked twice, a popup dialog is triggered and the data model is passed to
the dialog.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*******************************************************************************
***********/

public class CustomUINCountTableEditor extends JTextField implements
SimPopupTableEditor {
    private static final long serialVersionUID = -4236326754891359417L;

    private PopupTableEditorListener listener;
    private SimTableEditorEventAdaptor eventAdaptor;
    private Object model;


/*******************************************************************************
***********
 * Constructors

*******************************************************************************
**********/

    public CustomUINCountTableEditor(PopupTableEditorListener listener) {
        eventAdaptor = new SimTableEditorEventAdaptor(this);
```

```
        addMouseListener(buildMouseListener());
        setHorizontalAlignment(RIGHT);
        setOpaque(true);
        setEnabled(false);
        setListener(listener);
    }


/*******************************************************************************
**********
 * Basic Property Methods of a Table Editor

*******************************************************************************
********/

    public Class getValueClass() {
        return Integer.class;
    }

    public void setValueClass(Class valueClass) {
        // Ignore
    }

    public void setModel(Object model) {
        this.model = model;
    }

    public JComponent getComponent() {
        return this;
    }

    private void setListener(PopupTableEditorListener listener) {
        this.listener = listener;
    }

    public void setCoordinates(int row, int column) {
        // Ignore
    }


/*******************************************************************************
*********
 * Get, Set and Check the data value of the editor

*******************************************************************************
********/

    public Object getValue() {
        return getText();
    }

    public void setValue(Object value) {
        if (value != null) {
            setText(value.toString());
            popupDialog();
        }
    }

    public boolean checkValue() {
        return true;
```

```
    }


/*****************************************************************************
*************
 * Table Editor Listener

*****************************************************************************
***********/

    public void addTableEditorListener(SimTableEditorListener listener) {
        eventAdaptor.addTableEditorListener(listener);
    }

    public void removeTableEditorListener(SimTableEditorListener listener) {
        eventAdaptor.removeTableEditorListener(listener);
    }


/*****************************************************************************
************
 * Determines if keystroke is invalid for field. Always false for a boolean
editor.

*****************************************************************************
***********/
    public boolean isInvalidKeystroke(KeyEvent event) {
        return false;
    }


/*****************************************************************************
************
 * Mouse Listener That Pops The Dialog

*****************************************************************************
***********/
    private MouseListener buildMouseListener() {
        return new MouseAdapter() {
            public void mouseClicked(MouseEvent event) {
                if (event.getClickCount() == 2) {
                    popupDialog();
                }
            }
        };
    }


/*****************************************************************************
*************
 * Display Dialog

*****************************************************************************
***********/

    private void popupDialog() {
        if (listener != null) {
            listener.popupDialog(model);

            CustomUINCreateWrapper wrapper = (CustomUINCreateWrapper) model;
```

```
            setText(String.valueOf(wrapper.getSerialNumberCount()));
        }
    }
}
```

Example F–4 CustomUINCreateDialog

```
package oracle.retail.sim.shared.swing.custom;

import java.awt.GridBagLayout;
import java.util.Collections;
import java.util.List;
import java.util.Set;
import oracle.retail.sim.closed.application.Application;
import oracle.retail.sim.closed.common.ErrorKey;
import oracle.retail.sim.closed.simclient.util.SimClientErrorKey;
import oracle.retail.sim.closed.swing.dialog.RDialog;
import oracle.retail.sim.closed.swing.displayer.AttributeDisplayer;
import oracle.retail.sim.closed.swing.editor.RDisplayLabelEditor;
import oracle.retail.sim.closed.swing.event.RActionEvent;
import oracle.retail.sim.closed.swing.event.REventListener;
import oracle.retail.sim.closed.swing.panel.RPanel;
import oracle.retail.sim.closed.swing.table.SimTable;
import oracle.retail.sim.closed.swing.table.SimTableAttribute;
import oracle.retail.sim.closed.swing.table.SimTableDefinition;
import oracle.retail.sim.closed.swing.table.SimTableEditorEvent;
import oracle.retail.sim.closed.swing.table.SimTableEditorListener;
import oracle.retail.sim.closed.swing.table.SimTablePane;
import oracle.retail.sim.closed.swing.table.SimTableSortAttribute;
import oracle.retail.sim.closed.swing.util.GridTool;
import oracle.retail.sim.closed.swing.util.UIException;
import oracle.retail.sim.closed.swing.widget.RButton;
import oracle.retail.sim.closed.uin.SerialNumberValue;
import oracle.retail.sim.closed.uin.UINStatus;
import oracle.retail.sim.closed.util.CollectionUtil;
import oracle.retail.sim.shared.swing.core.SimNavigation;
import oracle.retail.sim.shared.swing.uin.SerialNumberProperty;
import oracle.retail.sim.shared.swing.uin.SerialNumberWrapper;

/*******************************************************************************
*************
 * This dialog handles entering UIN information for line item from the main UIN
Create screen.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*******************************************************************************
***********/

public class CustomUINCreateDialog extends RDialog implements REventListener {
    private static final long serialVersionUID = -7885503433032637549L;

    private CustomUINCreateDialogModel model = new CustomUINCreateDialogModel();

    private RDisplayLabelEditor itemEditor = new RDisplayLabelEditor("Item");
    private RDisplayLabelEditor itemDescEditor = new RDisplayLabelEditor("Item
Description");
```

```
    private CustomUINCreateDialogTableEditor serialNumberEditor
                                                = new
CustomUINCreateDialogTableEditor();

    private SimTable serialNumberTable = new SimTable(new
CreateUINTableDefinition());
    private SimTablePane serialNumberTablePane = new
SimTablePane(serialNumberTable);

    private RButton doneButton = new RButton(SimNavigation.DONE);
    private RButton addButton = new RButton(SimNavigation.ADD);
    private RButton deleteButton = new RButton(SimNavigation.DELETE);
    private RButton cancelButton = new RButton(SimNavigation.CANCEL);


/**************************************************************************
***********
 * Build Dialog

**************************************************************************
**********/
    public CustomUINCreateDialog() {
        super(Application.getFrame());
        setStatusBarVisible(false);
        setTitle("UIN");
        setSize(550, 400);
        initializeWidgets();
        layoutContent();
        centerWindow();
    }

    private void initializeWidgets() {

serialNumberEditor.addTableEditorListener(buildSerialNumberValueListener());

        serialNumberTable.setMultipleRowSelectionMode();

        doneButton.registerAction(this, SimNavigation.DONE);
        addButton.registerAction(this, SimNavigation.ADD);
        deleteButton.registerAction(this, SimNavigation.DELETE);
        cancelButton.registerAction(this, SimNavigation.CANCEL);
    }

    private void layoutContent() {
        addButton(doneButton);
        addButton(addButton);
        addButton(deleteButton);
        addButton(cancelButton);

        RPanel headerPanel = new RPanel(new GridBagLayout());
        headerPanel.add(itemEditor, GridTool.constraints(0, 0, 1, 1, 1, 0, 0, 3,
3, 0, 0, 10));
        headerPanel.add(itemDescEditor, GridTool.constraints(1, 0, 1, 1, 1, 0, 0,
3, 3, 0, 0, 0));

        RPanel mainPanel = new RPanel(new GridBagLayout());
        mainPanel.add(headerPanel, GridTool.constraints(0, 0, 1, 1, 1, 0, 0, 3, 0,
0, 5, 0));
        mainPanel.add(serialNumberTablePane, GridTool.constraints(0, 1, 1, 1, 1,
1, 0, 3, 0, 0, 0, 0));
```

```
        setContentPane(mainPanel);
    }


/*******************************************************************************
*************
 * Build Dialog

*******************************************************************************
***********/

    public void setWrapper(CustomUINCreateWrapper wrapper) {
        model.setWrapper(wrapper);
        try {
            itemEditor.setData(model.getItemId());
            itemDescEditor.setData(model.getItemDescription());
            serialNumberTable.setRows(model.getSerialNumberWrappers());
        } catch (Throwable exception) {
            displayException(exception);
        }
    }


/*******************************************************************************
*************
 * Listen to serial number entry field in order to validate UIN entered

*******************************************************************************
***********/

    private SimTableEditorListener buildSerialNumberValueListener() {
        return new SimTableEditorListener() {
            public void performTableEditorEvent(SimTableEditorEvent event) {
                if (event.isFocusLostEvent()) {
                    Set<String> serialNumberSet = CollectionUtil.newHashSet();
                    List<SerialNumberWrapper> wrappers =
serialNumberTable.getAllRowData();
                    for (SerialNumberWrapper wrapper : wrappers) {
                        SerialNumberValue serialNumber =
wrapper.getSerialNumberValue();
                        if (serialNumber != null) {
                            if (serialNumberSet.contains(serialNumber.getUin())) {
                                serialNumberTable.removeRow(wrapper);
                                displayException(
                                       new UIException(ErrorKey.UIN_DUPLICATE_
ERROR_PC));
                                continue;
                            }
                            if (serialNumber.getStatus() != UINStatus.UNCONFIRMED)
{
                                serialNumberTable.removeRow(wrapper);
                                displayException(new UIException(
                                                        "Serial Number already
exists!"));
                                continue;
                            }
                            serialNumberSet.add(serialNumber.getUin());
                        }
                    }
```

```
                    // Temporary fix - only add the new row if the button panel
does not have
                    if (isMouseOverButton()) {
                        return;
                    }
                    try {
                        doAddRow();
                    } catch (Throwable exception) {
                        displayException(exception);
                    }
                }
            }
        };
    }


/*********************************************************************************
*************
 * Handle Actions

*********************************************************************************
***********/
    public void performActionEvent(RActionEvent event) {
        String command = event.getEventCommand();
        try {
            if (command.equals(SimNavigation.DONE)) {
                doDone();
            } else if (command.equals(SimNavigation.ADD)) {
                doAddRow();
            } else if (command.equals(SimNavigation.DELETE)) {
                doDeleteRow();
            } else if (command.equals(SimNavigation.CANCEL)) {
                doCancel();
            }
        } catch (Throwable exception) {
            displayException(exception);
        }
    }


/*********************************************************************************
*************
 * Add Action

*********************************************************************************
***********/

    private void doAddRow() {
        serialNumberTable.stopEditing();
        List<SerialNumberWrapper> wrappers = serialNumberTable.getAllRowData();
        for (SerialNumberWrapper wrapper : wrappers) {
            if (wrapper.getSerialNumberValue() == null) {
                return;
            }
        }
        serialNumberTable.addRow(model.createSerialNumberWrapper());
        serialNumberTable.editCellInLastRow(SerialNumberProperty.SERIAL_NUMBER);
    }
```

```
/*****************************************************************************
*************
 * Delete Action

*****************************************************************************
***********/

    private void doDeleteRow() throws Exception {
        List<SerialNumberWrapper> selectedWrappers =
serialNumberTable.getAllSelectedRowData();
        if (selectedWrappers.isEmpty()) {
            throw new UIException(SimClientErrorKey.NO_ROWS_SELECTED_DELETE);
        }
        for (SerialNumberWrapper wrapper : selectedWrappers) {
            serialNumberTable.removeRow(wrapper);
        }
        serialNumberTable.refreshTable();
    }


/*****************************************************************************
*************
 * Done Action

*****************************************************************************
***********/

    private void doDone() throws Exception {
        serialNumberTable.stopEditing();
        List<SerialNumberWrapper> wrappers = serialNumberTable.getAllRowData();
        model.saveSerialNumbers(wrappers);
        closeWindow();
    }


/*****************************************************************************
*************
 * Override method to alter functionality. Default stops editing and closes
window.

*****************************************************************************
***********/
    private void doCancel() {
        serialNumberTable.stopEditing();
        closeWindow();
    }


/*****************************************************************************
*************
 * UIN Table Definition

*****************************************************************************
***********/

    private class CreateUINTableDefinition extends SimTableDefinition {

        public Class getDataClass() {
            return SerialNumberWrapper.class;
        }
```

```
        public List<SimTableSortAttribute> getSortAttributes() {
            return Collections.singletonList(new
SimTableSortAttribute("serialNumberValue"));
        }

        public List<SimTableAttribute> getAttributes() {
            List<SimTableAttribute> attributes = CollectionUtil.newArrayList(1);
            attributes.add(new SimTableAttribute(model.getUINLabel(),
"serialNumberValue",
                                new AttributeDisplayer("uin"), serialNumberEditor));
            return attributes;
        }
    }
}
```

Example F–5 CustomUINCreateDialogModel

```
package oracle.retail.sim.shared.swing.custom;

import java.util.List;
import oracle.retail.sim.closed.business.FunctionalArea;
import oracle.retail.sim.closed.uin.SerialNumberValue;
import oracle.retail.sim.closed.uin.UINType;
import oracle.retail.sim.closed.util.CollectionUtil;
import oracle.retail.sim.closed.util.RkConfigManager;
import oracle.retail.sim.shared.swing.core.SimScreenModel;
import oracle.retail.sim.shared.swing.uin.SerialNumberWrapper;

/*******************************************************************************
*************
 * The business logic model for the Custom UIN Create Dialog.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*******************************************************************************
***********/

public class CustomUINCreateDialogModel extends SimScreenModel {

    private CustomUINCreateWrapper lineItemWrapper;

    public void setWrapper(CustomUINCreateWrapper wrapper) {
        lineItemWrapper = wrapper;
    }

    public String getItemId() {
        if (lineItemWrapper != null) {
            return lineItemWrapper.getStockItem().getId();
        }
        return null;
    }

    public String getItemDescription() {
        if (lineItemWrapper != null) {
            if (RkConfigManager.isItemShortDescription()) {
                return lineItemWrapper.getStockItem().getShortDescription();
            }
```

```
            return lineItemWrapper.getStockItem().getLongDescription();
        }
        return null;
    }

    public List<SerialNumberWrapper> getSerialNumberWrappers() {
        List<SerialNumberWrapper> wrappers = CollectionUtil.newArrayList();
        for (SerialNumberValue value : lineItemWrapper.getSerialNumbers()) {
            SerialNumberWrapper wrapper = createSerialNumberWrapper();
            wrapper.setSerialNumberValue(value);
            wrapper.setDefaultAction();
            wrapper.setValidated();

            wrappers.add(wrapper);
        }
        if (wrappers.isEmpty()) {
            wrappers.add(createSerialNumberWrapper());
        }
        return wrappers;
    }

    public SerialNumberWrapper createSerialNumberWrapper() {
        return new SerialNumberWrapper(FunctionalArea.MANUAL,
                  lineItemWrapper.getStockItem().getId(), getUINType(),
getUINLabel());
    }

    public UINType getUINType() {
        if (lineItemWrapper != null) {
            return lineItemWrapper.getStockItem().getUINType();
        }
        return UINType.SERIAL;
    }

    public String getUINLabel() {
        if (lineItemWrapper != null) {
            return lineItemWrapper.getStockItem().getUINLabel();
        }
        return null;
    }

    public void saveSerialNumbers(List<SerialNumberWrapper> wrappers) {
        List<SerialNumberValue> serialNumbers = CollectionUtil.newArrayList();
        for (SerialNumberWrapper wrapper : wrappers) {
            SerialNumberValue value = wrapper.getSerialNumberValue();
            if (value != null) {
                serialNumbers.add(value);
            }
        }
        lineItemWrapper.setSerialNumbers(serialNumbers);
    }
}
```

Example F–6 CustomUINCreateDialogTableEditor

```
package oracle.retail.sim.shared.swing.custom;

import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyEvent;
```

```
import javax.swing.JComponent;
import oracle.retail.sim.closed.application.Application;
import oracle.retail.sim.closed.business.ClientServiceFactory;
import oracle.retail.sim.closed.business.FunctionalArea;
import oracle.retail.sim.closed.common.BusinessException;
import oracle.retail.sim.closed.common.ErrorKey;
import oracle.retail.sim.closed.common.locale.StringConstants;
import oracle.retail.sim.closed.common.type.AbstractDisplayer;
import oracle.retail.sim.closed.locale.StringUtility;
import oracle.retail.sim.closed.locale.Translator;
import oracle.retail.sim.closed.simclient.util.SimClientErrorKey;
import oracle.retail.sim.closed.swing.dialog.RErrorDialog;
import oracle.retail.sim.closed.swing.displayer.AttributeDisplayer;
import oracle.retail.sim.closed.swing.logging.UILog;
import oracle.retail.sim.closed.swing.sim.SimRepository;
import oracle.retail.sim.closed.swing.table.SimTable;
import oracle.retail.sim.closed.swing.table.SimTableEditor;
import oracle.retail.sim.closed.swing.table.SimTableEditorEventAdaptor;
import oracle.retail.sim.closed.swing.table.SimTableEditorListener;
import oracle.retail.sim.closed.swing.util.UIErrorKey;
import oracle.retail.sim.closed.swing.util.UIExceptionFactory;
import oracle.retail.sim.closed.swing.widget.RTextField;
import oracle.retail.sim.closed.uin.SerialNumberValue;
import oracle.retail.sim.shared.swing.core.SimName;
import oracle.retail.sim.shared.swing.uin.SerialNumberWrapper;

/*****************************************************************************
*************
 * Table Editor for editing a CustomerSerialNumber business object. The model for
the table
 * row that uses this editor
 * must be a UINWrapper.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*****************************************************************************
***********/

public class CustomUINCreateDialogTableEditor extends RTextField
                                              implements SimTableEditor,
FocusListener {
    private static final long serialVersionUID = -1463881160730583097L;

    private AbstractDisplayer displayer;
    private SimTableEditorEventAdaptor eventAdaptor;
    private SerialNumberWrapper wrapper;
    private SerialNumberValue serialNumberValue;
    private String lastCheckedSerialNumber = StringConstants.EMPTY;
    private FocusEvent lastFocusEvent;
    private boolean isErrorState;
    private int row = -1;
    private int column = -1;

    public CustomUINCreateDialogTableEditor() {
        eventAdaptor = new SimTableEditorEventAdaptor(this);
        displayer = new AttributeDisplayer("uin");
        setIdentifier(SimName.SERIAL_NUMBER);
        setMargin(null);
```

```
        setBorder(null);
        addFocusListener(this);
    }

    public Class getValueClass() {
        return SerialNumberValue.class;
    }

    public void setValueClass(Class valueClass) {
        // Ignore
    }

    public void setModel(Object model) {
        wrapper = (SerialNumberWrapper) model;
    }


/*******************************************************************************
*************
 * Assign coordinates to the editor.

*******************************************************************************
***********/
    public void setCoordinates(int row, int column) {
        this.row = row;
        this.column = column;
    }


/*******************************************************************************
*************
 * Reactivate editing within the table cell.

*******************************************************************************
***********/
    private void reactivateEditing(Object object) {
        if (object instanceof SimTable) {
            SimTable table = (SimTable) object;
            try {
                if (table.getSelectedRow() != row) {
                    table.setRowSelectionInterval(row, row);
                }
                table.editCellAt(row, column);
            } catch (Throwable ex) {
                UILog.debug(getClass(), ex.getMessage());
            }
        }
    }

    public Object getValue() {
        if (isErrorState()) {
            return null;
        }
        return serialNumberValue;
    }

    public void setData(Object value) {
        if (value instanceof SerialNumberValue) {
            setValue(value);
            return;
```

```
        }
        throw new IllegalArgumentException(
                                "CustomSerialNumberTableEditor only edits
UINValue!");
    }

    public void setValue(Object value) {
        if (value instanceof String) {
            setText((String) value);
            checkValue();
        } else if (value instanceof SerialNumberValue) {
            serialNumberValue = (SerialNumberValue) value;
            setText(displayer.getDisplayText(serialNumberValue));
        } else if (value == null) {
            serialNumberValue = null;
            clear();
        }
        eventAdaptor.fireTypeEditorEvent();
    }

    public JComponent getComponent() {
        return this;
    }

    public boolean checkValue() {
        String enteredSerialNumber = getText();

        setErrorState(false);

        if (StringUtility.isNullOrEmpty(enteredSerialNumber)) {
            if (serialNumberValue == null) {
                return true;
            }
            displayErrorWithReset(SimClientErrorKey.ITEM_BLANK_ERROR);
            return false;
        }

        try {
            if (serialNumberValue != null
               && enteredSerialNumber.equals(serialNumberValue.getUin())
               && enteredSerialNumber.equals(lastCheckedSerialNumber)) {
                return true;
            }
            SerialNumberValue tempValue =

ClientServiceFactory.getUINServices().findSerialNumberValueOrCreate(
                     SimRepository.getStoreId(), wrapper.getItemId(),
enteredSerialNumber,
                    wrapper.getType(), FunctionalArea.MANUAL);
            if (tempValue == null) {
                String message = Translator.getMessage(ErrorKey.UIN_NOT_FOUND_FOR_
ITEM,
                 wrapper.getType().getDescription(), enteredSerialNumber,
wrapper.getItemId());
                displayErrorWithReset(message);
                return false;
            }
            serialNumberValue = tempValue;
            lastCheckedSerialNumber = enteredSerialNumber;
            return true;
```

```
        } catch (BusinessException businessException) {

displayError(UIExceptionFactory.buildError(businessException).getLocalizedMessage(
));
        } catch (Throwable t) {
            displayError(UIErrorKey.DEFAULT_FATAL_DIALOG_MESSAGE);
        }
        lastCheckedSerialNumber = StringConstants.EMPTY;
        serialNumberValue = null;
        clear();
        return false;
    }


/**************************************************************************************
*************
 * Add Remove Table Editor Listeners

**************************************************************************************
***********/

    public void addTableEditorListener(SimTableEditorListener listener) {
        eventAdaptor.addTableEditorListener(listener);
    }

    public void removeTableEditorListener(SimTableEditorListener listener) {
        eventAdaptor.removeTableEditorListener(listener);
    }


/**************************************************************************************
************
 * Error State

**************************************************************************************
***********/
    protected void setErrorState(boolean errorState) {
        isErrorState = errorState;
    }

    protected boolean isErrorState() {
        return isErrorState;
    }


/**************************************************************************************
*************
 * Determines if keystroke is invalid for input into the field.

**************************************************************************************
***********/
    public boolean isInvalidKeystroke(KeyEvent event) {
        return false;
    }


/**************************************************************************************
************
 * Implement the focus listener methods to call do value modified when focus is
lost.
```

```
*****************************************************************************
**********/
    public void focusGained(FocusEvent event) {
    }

    public void focusLost(FocusEvent event) {
        if (event.isTemporary()) {
            return;
        }
        if (event != lastFocusEvent) {
            lastFocusEvent = event;
            if (checkValue()) {
                eventAdaptor.fireTypeEditorEvent(true);
                return;
            }
            reactivateEditing(event.getOppositeComponent());
        }
    }


/*****************************************************************************
*************
 * Display Error Methods

*****************************************************************************
**********/

    protected void displayErrorWithReset(String message) {
        if (displayer != null) {
            setText(displayer.getDisplayText(serialNumberValue));
        }
        setErrorState(true);
        displayError(message);
    }

    protected void displayError(String message) {
        RErrorDialog dialog = new RErrorDialog(Application.getFrame());
        dialog.setTitle("Table Editor Error");
        dialog.setMessage(message);
        dialog.activate();
    }
}
```

Example F–7 CustomUINCreateModel

```
package oracle.retail.sim.shared.swing.custom;

import java.util.List;
import oracle.retail.sim.closed.application.RepositoryManager;
import oracle.retail.sim.closed.custom.CustomSerialNumber;
import oracle.retail.sim.closed.custom.CustomUINEJBServices;
import oracle.retail.sim.closed.uin.SerialNumberValue;
import oracle.retail.sim.closed.util.CollectionUtil;
import oracle.retail.sim.shared.swing.core.SimScreenModel;

/*****************************************************************************
*************
 * Model for the Create UIN entry screen.
 * <p>
```

```
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*****************************************************************************
***********/

public class CustomUINCreateModel extends SimScreenModel {

    private String CUSTOM_UIN_SERVICES_KEY = "CUSTOM_UIN_SERVICES_KEY";

    public void saveSerialNumbers(List<CustomUINCreateWrapper> wrappers) throws
Exception {
        List<CustomSerialNumber> serialNumbers = CollectionUtil.newArrayList();
        for (CustomUINCreateWrapper wrapper : wrappers) {
            for (SerialNumberValue serialNumberValue : wrapper.getSerialNumbers())
{
                CustomSerialNumber serialNumber = new CustomSerialNumber();
                serialNumber.setId(serialNumberValue.getUINDetailId());
                serialNumber.setStatus(serialNumberValue.getStatus());
                serialNumbers.add(serialNumber);
            }
        }
        getCustomUINService().moveSerialNumbersToInStock(serialNumbers);
    }

    private CustomUINEJBServices getCustomUINService() {
        CustomUINEJBServices services = (CustomUINEJBServices)
                        RepositoryManager.getStateObject(CUSTOM_UIN_SERVICES_
KEY);
        if (services == null) {
            services = new CustomUINEJBServices();
            RepositoryManager.addStateObject(CUSTOM_UIN_SERVICES_KEY, services);
        }
        return services;
    }
}
```

Example F–8 CustomUINCreatePanel

```
package oracle.retail.sim.shared.swing.custom;

import java.util.Collections;
import java.util.List;
import java.util.Set;
import oracle.retail.sim.closed.item.StockItem;
import oracle.retail.sim.closed.swing.displayer.AttributeDisplayer;
import oracle.retail.sim.closed.swing.displayer.IntegerDisplayer;
import oracle.retail.sim.closed.swing.event.RActionEvent;
import oracle.retail.sim.closed.swing.event.REventListener;
import oracle.retail.sim.closed.swing.frame.ScreenPanel;
import oracle.retail.sim.closed.swing.table.SimTable;
import oracle.retail.sim.closed.swing.table.SimTableAttribute;
import oracle.retail.sim.closed.swing.table.SimTableDefinition;
import oracle.retail.sim.closed.swing.table.SimTableEditorEvent;
import oracle.retail.sim.closed.swing.table.SimTableEditorListener;
import oracle.retail.sim.closed.swing.table.SimTablePane;
import oracle.retail.sim.closed.swing.tableeditor.PopupTableEditorListener;
import oracle.retail.sim.closed.util.CollectionUtil;
import oracle.retail.sim.shared.swing.uom.StockItemTableEditor;
```

```
/*****************************************************************************
*************
 * UIN Create Panel - Contains the visual elements of the UIN Create screen.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*****************************************************************************
***********/

public class CustomUINCreatePanel extends ScreenPanel implements REventListener {
    private static final long serialVersionUID = 7667002106326748099L;

    private CustomUINCreateModel model = new CustomUINCreateModel();

    private StockItemTableEditor stockItemTableEditor = new
StockItemTableEditor();

    private SimTable stockItemTable = new SimTable(new
CustomCreateUINTableDefinition());
    private SimTablePane stockItemPane = new SimTablePane(stockItemTable);


/*****************************************************************************
************
 * Initialize Panel

*****************************************************************************
***********/

    public CustomUINCreatePanel() {
        initializePanel();
        layoutPanel();
    }

    private void initializePanel() {
        stockItemTableEditor.addTableEditorListener(buildStockItemListener());
        stockItemTable.setColumnSize("serialNumberCount", SimTable.LABEL_WIDTH);
    }

    private void layoutPanel() {
        setContentPane(stockItemPane);
    }

    public boolean isStartable() {
        return true;
    }

    public void start() throws Throwable {
        stockItemTable.addRow(new CustomUINCreateWrapper());
    }

    public void stop() {
    }

    public void performActionEvent(RActionEvent event) {
    }
```

```
/*******************************************************************************
***********
 * Handle Add Item Action

*******************************************************************************
**********/

    public void doAddItem() {
        List<CustomUINCreateWrapper> wrappers = stockItemTable.getAllRowData();
        for (CustomUINCreateWrapper wrapper : wrappers) {
            if (wrapper.getStockItem() == null) {
                return;
            }
        }
        stockItemTable.addRow(new CustomUINCreateWrapper());
    }


/*******************************************************************************
************
 * Handle Delete Item Action

*******************************************************************************
***********/

    public void doDeleteItem() {
        List<CustomUINCreateWrapper> wrappers =
stockItemTable.getAllSelectedRowData();
        for (CustomUINCreateWrapper wrapper : wrappers) {
            stockItemTable.removeRow(wrapper);
        }
    }


/*******************************************************************************
************
 * Handle Done Action

*******************************************************************************
***********/

    public void doHandleDone() throws Exception {
        List<CustomUINCreateWrapper> wrappers = stockItemTable.getAllRowData();
        model.saveSerialNumbers(wrappers);
    }


/*******************************************************************************
************
 * Stock Item Listener - Accessed when stock item is added to table of items.

*******************************************************************************
***********/

    private SimTableEditorListener buildStockItemListener() {
        return new SimTableEditorListener() {
            public void performTableEditorEvent(SimTableEditorEvent event) {
                Set<StockItem> stockItems = CollectionUtil.newHashSet();
                List<CustomUINCreateWrapper> wrappers =
```

```
stockItemTable.getAllSelectedRowData();
                for (CustomUINCreateWrapper wrapper : wrappers) {
                    if (stockItems.contains(wrapper.getStockItem())) {
                        displayError("Duplicate item has been added.");
                        stockItemTable.removeRow(wrapper);
                        return;
                    }
                    stockItems.add(wrapper.getStockItem());
                }
            }
        };
    }


/*********************************************************************************
************
 * Table Definition

*********************************************************************************
***********/

    private class CustomCreateUINTableDefinition extends SimTableDefinition {

        public Class getDataClass() {
            return CustomUINCreateWrapper.class;
        }

        public List<String> getOverrideEditableAttributes() {
            return Collections.singletonList(CustomUINCreateWrapper.UIN_COUNT_
PROPERTY);
        }

        public List<SimTableAttribute> getAttributes() {
            List<SimTableAttribute> attributes = CollectionUtil.newArrayList(3);
            attributes.add(new SimTableAttribute("Item",
                    CustomUINCreateWrapper.STOCK_ITEM_PROPERTY, new
AttributeDisplayer("id"),
                    stockItemTableEditor));
            attributes.add(new SimTableAttribute("Item Description",
                    CustomUINCreateWrapper.DESCRIPTION_PROPERTY));
            attributes.add(new SimTableAttribute("UIN Qty",
                    CustomUINCreateWrapper.UIN_COUNT_PROPERTY, new
IntegerDisplayer(),
                    new CustomUINCountTableEditor(new UINPopupListener())));
            return attributes;
        }
    }


/*********************************************************************************
************
 * UIN POPUP LISTENER - Pops up when UIN column is double-clicked.

*********************************************************************************
***********/

    private class UINPopupListener implements PopupTableEditorListener {
        public void popupDialog(Object lineItem) {
            CustomUINCreateDialog dialog = new CustomUINCreateDialog();
            dialog.setWrapper((CustomUINCreateWrapper) lineItem);
```

```
            dialog.setVisible(true);
            stockItemTable.setRows(stockItemTable.getAllRowData());
        }
    }
}
```

Example F–9 CustomUINCreateScreen

```
package oracle.retail.sim.shared.swing.custom;

import oracle.retail.sim.closed.application.NavigationEvent;
import oracle.retail.sim.closed.swing.sim.SimScreen;
import oracle.retail.sim.shared.swing.core.SimNavigation;

/*******************************************************************************
*************
 * UIN Create Screen - This screens provides the ability to create and insert new
UINs into
 * SIM without validation or generating matching transactions. This should be used
exclusively
 * to dataseed UINs into the application.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*******************************************************************************
***********/

public class CustomUINCreateScreen extends SimScreen {
    private static final long serialVersionUID = 487712289865375658L;

    private CustomUINCreatePanel panel = new CustomUINCreatePanel();

    public CustomUINCreateScreen() {
        add(panel);
    }


/*******************************************************************************
*************
 * Screen Methods

*******************************************************************************
***********/

    public String getScreenName() {
        return "UIN Create";
    }

    public boolean isStartable() {
        return panel.isStartable();
    }

    public void start() throws Throwable {
        showMenu();
        panel.start();
    }

    public void stop() {
```

```
        panel.stop();
    }


/*****************************************************************************
*************
 * Handle Navigation Events

*****************************************************************************
***********/

    public void navigationEvent(NavigationEvent event) {
        String command = event.getCommand();
        try {
            if (command.equals(SimNavigation.ADD_ITEM)) {
                panel.doAddItem();
            } else if (command.equals(SimNavigation.DELETE_ITEM)) {
                panel.doDeleteItem();
            } else if (command.equals(SimNavigation.DONE)) {
                panel.doHandleDone();
            }
        } catch (Throwable exception) {
            displayException(panel, event, exception);
        }
    }
}
```

Example F–10 CustomUINCreateWrapper

```
package oracle.retail.sim.shared.swing.custom;

import java.util.List;
import oracle.retail.sim.closed.common.BusinessException;
import oracle.retail.sim.closed.common.locale.StringConstants;
import oracle.retail.sim.closed.item.StockItem;
import oracle.retail.sim.closed.uin.SerialNumberValue;
import oracle.retail.sim.closed.util.CollectionUtil;
import oracle.retail.sim.closed.util.RkConfigManager;

/*****************************************************************************
*************
 * Wrapper for single entry row in the serial number table.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*****************************************************************************
***********/

public class CustomUINCreateWrapper {

    public static final String STOCK_ITEM_PROPERTY = "stockItem";
    public static final String DESCRIPTION_PROPERTY = "description";
    public static final String UIN_COUNT_PROPERTY = "serialNumberCount";

    private StockItem stockItem;
    private List<SerialNumberValue> serialNumbers = CollectionUtil.newArrayList();

    public StockItem getStockItem() {
```

```
        return stockItem;
    }

    public String getDescription() {
        if (stockItem == null) {
            return StringConstants.EMPTY;
        }
        if (RkConfigManager.isItemShortDescription()) {
            return stockItem.getShortDescription();
        }
        return stockItem.getLongDescription();
    }

    public void setStockItem(StockItem stockItem) throws BusinessException {
        if (stockItem != null && !stockItem.isSerialNumberRequired()) {
            throw new BusinessException("Item does not support serial numbers.");
        }
        this.stockItem = stockItem;
    }

    public Integer getSerialNumberCount() {
        return serialNumbers.size();
    }

    public List<SerialNumberValue> getSerialNumbers() {
        return serialNumbers;
    }

    public void setSerialNumbers(List<SerialNumberValue> serialNumbers) {
        if (serialNumbers != null) {
            this.serialNumbers = serialNumbers;
        }
    }

    public boolean isPropertyModifiable(String property) {
        if (property.equals(STOCK_ITEM_PROPERTY)) {
            return stockItem == null;
        }
        if (property.equals(UIN_COUNT_PROPERTY)) {
            return true;
        }
        return false;
    }
}
```

Example F–11 CustomUINDataBean

```
package oracle.retail.sim.shared.dataaccess.databean.generated;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.ArrayList;
import java.util.List;
import oracle.retail.sim.closed.dataaccess.FullDataBean;

/**
 *
 * This class is an object representation of the database table UIN_DETAIL<BR>
 * Followings are the column of the table: <BR>
```

```
 *     ID(PK) -- NUMERIC(12)<BR>
 *     STORE_ID -- NUMERIC(10)<BR>
 *     STATUS -- NUMERIC(2)<BR>
 *
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
public class CustomUinDataBean extends FullDataBean<CustomUinDataBean> {
    private static String getSelectSqlText() {
        return "select ID, STORE_ID, STATUS from UIN_DETAIL ";
    }

    private static String getInsertSqlText() {
        return "insert into UIN_DETAIL (ID, STORE_ID, STATUS) values (?, ?, ?)";
    }

    private static String getUpdateSqlText() {
        return "update UIN_DETAIL set STORE_ID = ?, STATUS = ? ";
    }

    private static String getDeleteSqlText() {
        return "delete from UIN_DETAIL ";
    }

    public static final String TABLE_NAME = "UIN_DETAIL";
    public static final String COL_ID = "UIN_DETAIL.ID";
    public static final String COL_STORE_ID = "UIN_DETAIL.STORE_ID";
    public static final String COL_STATUS = "UIN_DETAIL.STATUS";

    public static final int TYP_ID = Types.NUMERIC;
    public static final int TYP_STORE_ID = Types.NUMERIC;
    public static final int TYP_STATUS = Types.NUMERIC;

    public static final String SELECT_SQL = getSelectSqlText();
    public static final String INSERT_SQL = getInsertSqlText();
    public static final String UPDATE_SQL = getUpdateSqlText();
    public static final String DELETE_SQL = getDeleteSqlText();

    private Long id;
    private Long storeId;
    private Long status;

    public CustomUinDataBean() {
    }

    public String getSelectSql() {
        return SELECT_SQL;
    }

    public String getInsertSql() {
        return INSERT_SQL;
    }

    public String getUpdateSql() {
        return UPDATE_SQL;
    }

    public String getDeleteSql() {
        return DELETE_SQL;
    }
```

```
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public Long getStoreId() {
        return storeId;
    }

    public void setStoreId(Long storeId) {
        this.storeId = storeId;
    }

    public void setStoreId(long storeId) {
        this.storeId = storeId;
    }

    public Long getStatus() {
        return status;
    }

    public void setStatus(Long status) {
        this.status = status;
    }

    public void setStatus(long status) {
        this.status = status;
    }

    public CustomUinDataBean read(ResultSet resultSet) throws SQLException {
        CustomUinDataBean bean = new CustomUinDataBean();
        bean.id = getLongFromResultSet(resultSet, "ID");
        bean.storeId = getLongFromResultSet(resultSet, "STORE_ID");
        bean.status = getLongFromResultSet(resultSet, "STATUS");
        return bean;
    }

    public List<Object> toList(boolean includePrimaryKey) {
        List<Object> list = new ArrayList<Object>();
        if (includePrimaryKey) {
          addToList(list, id, TYP_ID);
        }
        addToList(list, storeId, TYP_STORE_ID);
        addToList(list, status, TYP_STATUS);
        return list;
    }
}
```

Example F–12 CustomUINEJBServices

```
package oracle.retail.sim.closed.custom;

import oracle.retail.sim.closed.common.BusinessException;
import oracle.retail.sim.closed.common.DowntimeException;
import oracle.retail.sim.closed.common.SimServerException;
import oracle.retail.sim.closed.ejb.CustomUINInterface;
import oracle.retail.sim.closed.logging.LogNames;
import oracle.retail.sim.closed.logging.LogService;
import oracle.retail.sim.closed.util.CompressedObject;
import oracle.retail.sim.closed.util.EJBServicesManager;
import oracle.retail.sim.closed.util.SimObjectUtils;
import oracle.retail.sim.closed.util.UniversalContext;

/**
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
public class CustomUINEJBServices extends CustomUINServices {
    private CustomUINInterface lookup() throws Exception {
        try {
            return (CustomUINInterface)
EJBServicesManager.cachedLookup("CustomUINBean");
        } catch (Throwable t) {
            throw new DowntimeException(
"An error occurred accessing CustomUINServices. Please contact your system
administrator.", t);
        }
    }

    private void removeCache() {
        EJBServicesManager.removeCache("CustomUINBean");
    }

    public void moveSerialNumbersToInStock(

java.util.List<oracle.retail.sim.closed.custom.CustomSerialNumber> arg0)
                    throws Exception {
        Throwable causeOfException = null;
        int maxAttempts = EJBServicesManager.getMaxConnectAttempts();
        CompressedObject compressedSimSession =
                      new CompressedObject(UniversalContext.getSession());
        CompressedObject compressed0 = new CompressedObject(arg0);
        if (LogService.isDebugEnabled(LogNames.SERIALIZED_OBJECT_SIZES)) {
            LogService.debug(LogNames.SERIALIZED_OBJECT_SIZES,
"CustomUINBean.moveSerialNumbersToInStock(compressed0, compressedSimSession)
(remote call) is sending " + SimObjectUtils.calculateByteSize(compressed0,
compressedSimSession) + " bytes in serialized object(s) for the remote call.");
        }
        for (int i = 0; i < maxAttempts; i++) {
            CustomUINInterface bean = lookup();
            try {
                long startTime = System.currentTimeMillis();
                CompressedObject compressedReturnVal =
                       bean.moveSerialNumbersToInStock(compressed0,
compressedSimSession);
                long endTime = System.currentTimeMillis();
                if (LogService.isDebugEnabled(LogNames.SERVICE_TIMINGS)) {
                    LogService.debug(LogNames.SERVICE_TIMINGS,
"CustomUINBean.moveSerialNumbersToInStock(compressed0, compressedSimSession)
(remote call) took " + (endTime - startTime) + "ms to complete");
```

```
                }
                if (LogService.isDebugEnabled(LogNames.SERIALIZED_OBJECT_SIZES)) {
                    LogService.debug(LogNames.SERIALIZED_OBJECT_SIZES,
"CustomUINBean.moveSerialNumbersToInStock(compressed0, compressedSimSession)
(remote call) received " + SimObjectUtils.calculateByteSize(compressedReturnVal) +
" bytes in the returned serialized object.");
                }
                return;
            } catch (SimServerException sse) {
                throw sse;
            } catch (BusinessException be) {
                throw be;
            } catch (Throwable t) {
                causeOfException = t;
                LogService.error(this, "Unexpected error in EJB connection
attempt: " + i, t);
                removeCache();
            }
        }
        throw new DowntimeException("An error occurred accessing
CustomUINServices. Please contact your system administrator.", causeOfException);
    }

}
```

Example F–13 CustomUINInterface

```
package oracle.retail.sim.closed.ejb;

import javax.ejb.Remote;
import oracle.retail.sim.closed.util.CompressedObject;

/**
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
@Remote
public interface CustomUINInterface {
    CompressedObject moveSerialNumbersToInStock(CompressedObject compressed0,
CompressedObject compressedSimSession) throws Exception;

}
```

Example F–14 CustomUINOracleDAO

```
package oracle.retail.sim.shared.dataaccess.dao;

import java.util.List;
import oracle.retail.sim.closed.common.SimServerException;
import oracle.retail.sim.closed.custom.CustomSerialNumber;
import oracle.retail.sim.closed.dataaccess.BaseOracleDao;
import oracle.retail.sim.closed.dataaccess.BatchParametricStatement;
import oracle.retail.sim.closed.uin.UINStatus;
import oracle.retail.sim.closed.util.CollectionUtil;
import oracle.retail.sim.shared.dataaccess.databean.generated.UinDetailDataBean;

/*******************************************************************************
*************
 * This class contains the database layer code for inserting new serial numbers
into
 * the database.
 * <p>
```

```
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.

*****************************************************************************
**********/
public class CustomUINOracleDao extends BaseOracleDao {

    public void saveSerialNumbers(Long storeId, List<CustomSerialNumber>
serialNumbers)
                                                                    throws
SimServerException {
        StringBuilder sql = new StringBuilder();
        sql.append(" UPDATE ").append(UinDetailDataBean.TABLE_NAME);
        sql.append("    SET ").append(UinDetailDataBean.COL_STATUS).append(" = ?
");
        sql.append("  WHERE ").append(UinDetailDataBean.COL_ID).append(" = ? ");
        sql.append("    AND ").append(UinDetailDataBean.COL_STORE_ID).append(" = ?
");
        sql.append("    AND ").append(UinDetailDataBean.COL_STATUS).append(" = ?
");

        BatchParametricStatement batchStatement = new
BatchParametricStatement(sql.toString());
        List<Object> params = null;

        for (CustomSerialNumber serialNumber : serialNumbers) {
            params = CollectionUtil.newArrayList(2);
            params.add(UINStatus.IN_STOCK.getCode());
            params.add(serialNumber.getId());
            params.add(storeId);
            params.add(serialNumber.getStatus().getCode());

            batchStatement.addParams(params);
        }
        executeBatch(batchStatement);
    }
}
```

Example F–15 CustomUINServerServices

```
package oracle.retail.sim.closed.custom;

import java.util.List;
import oracle.retail.sim.closed.util.UniversalContext;
import oracle.retail.sim.shared.dataaccess.dao.CustomUINOracleDao;

/**
 * Implements the Custom UIN services.
 * <p>
 * This is future work-in-progress code for future SIM 14.0 or later release.
 * <p>
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
public class CustomUINServerServices extends CustomUINServices {

    public void moveSerialNumbersToInStock(List<CustomSerialNumber> serialNumbers)
                                                                      throws
Exception {
        CustomUINOracleDao dao = new CustomUINOracleDao();
```

```
        dao.saveSerialNumbers(UniversalContext.getStoreId(), serialNumbers);
    }
}
```

Example F–16 CustomUINServices

```
package oracle.retail.sim.closed.custom;

import java.util.List;

/**
 * Defines the API for Custom UIN Services.
 *
 * Copyright 2004, 2011, Oracle. All rights reserved.
 */
public abstract class CustomUINServices {
    /**
     * This API will move all serial numbers include in the parameters from the
current status
     * to IN_STOCK. This will only happen if the serial number is currently in the
status
     * specified  in the parameter object. This will write a history record of the
movement.
     * @param serialNumbers The serial numbers to update.
     */
    public abstract void moveSerialNumbersToInStock(List<CustomSerialNumber>
serialNumbers) throws Exception;
}
```

G

Appendix: Data Seed UIN Attributes File Layout

The Data Seed UIN Attributes input file contains the following comma-separated values:

Table G–1 UpdateUINAttributes Input File Details

Record Name	Field Name	FieldType	Default Value	Description
1	Store ID	Number	NA	ID of the Store, for example, 1111.
2	Department ID	Number	NA	ID of the Department, for example, 2222.
3	Class ID	Number	NA	ID of the Class, for example, 2.
4	Type	Varchar2	NA	Type of UIN, for example, Serial Number.
5	Label	Varchar2	NA	Label of UIN, for example, Serial Number.
6	Capture_ Time_ID	Number	NA	Capture Time ID, for example, 2.
7	External_ Create_ Allowed	Varchar2	NA	External Create Allowed or not, for example, Y or N.

The following is a sample input file:

```
3000,3023,2,'Auto Generate SN','Serial Number',2,False
```

Index

T