Oracle® Retail Store Inventory Management

Implementation Guide
Release 13.0

April 2008

ORACLE

Oracle® Store Inventory Management Implementation Guide, Release 13.0

Primary Author: Graham Fredrickson

Value-Added Reseller (VAR) Language

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

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

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

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

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

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

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

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

(ix) the software component known as **WebLogic™** developed and licensed by BEA Systems, Inc. of San Jose, California, to Oracle and imbedded in the Oracle Retail Value Chain Collaboration application.

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

Contents

Preface

The Oracle Retail Store Inventory Management Implementation Guide provides detailed information that is important when implementing SIM.

The SIM Implementation Guide 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 customizations.
- 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

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

Related Documents

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

- Oracle Retail Store Inventory Management Data Model
- Oracle Retail Store Inventory Management Handheld Terminal Quick Reference Guide
- Oracle Retail Store Inventory Management 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

https://metalink.oracle.com

When contacting Customer Support, please provide the following:

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

Review Patch Documentation

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

Oracle Retail Documentation on the Oracle Technology Network

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

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

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

Conventions

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

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

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

A hyperlink appears like this.

1

Technical Architecture

Overview

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.

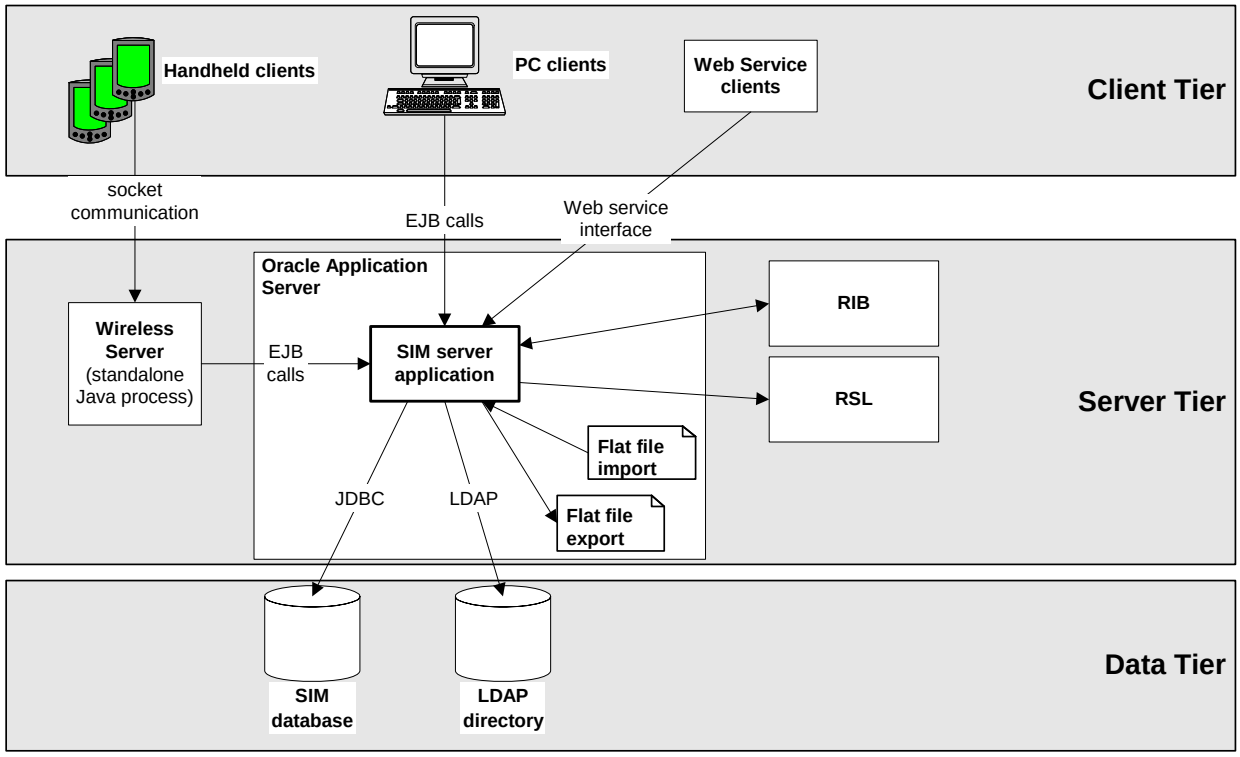
The n-tier architecture of SIM allows for the encapsulation of business logic, shielding the client from the complexity of the back-end system. Any given tier need not be concerned with the internal functional tasks of any other tier.

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 diagram below illustrates the major pieces of the typical three-tiered SIM implementation.



SIM's Technical Architecture

Client Tier

SIM can be deployed on a wide variety of clients, including a desktop computer, a hand-held wireless device, and so on. The 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 via the industry standard Java database connectivity (JDBC) protocol. In order for the SIM client to retrieve the desired data from the database, a JDBC connection must exist between the middle tier and the database. JDBC facilitates the communication between a Java application and a relational database. In essence, JDBC is a set of application programming interfaces (API)s that offer a database-independent means of extracting and/or inserting data to or from a database. To perform those insertions and extractions, SQL code also resides in this tier facilitating create, read, update, and delete actions.

Database Tier

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

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

Distributed Topology

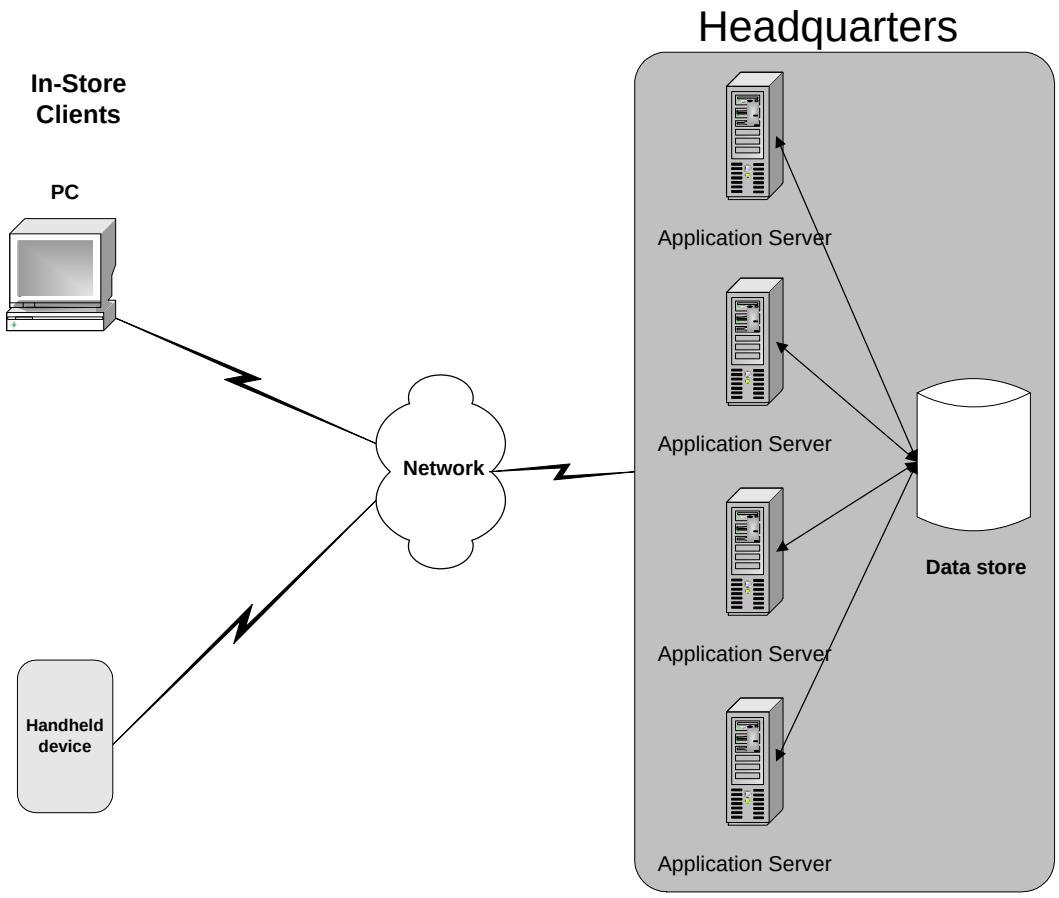
One of SIM's most significant advantages is its flexible distributed topology. SIM offers complete location transparency because the location of data and/or services is based upon the retailer's business requirements, not upon technical limitations. SIM's client server communication is an EJB call (which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

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

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

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

The following diagram illustrates SIM's deployment.



SIM's Deployment

Oracle Single Sign-on Overview

What is Single Sign-On?

Single Sign-On (SSO) is a term for the ability to sign onto multiple Web applications using a single user ID/Password. There are many implementations of SSO. Oracle currently provides three different implementations: Oracle Single Sign-On (OSSO), Java SSO (with the 10.1.3.1 release of OC4J) and Oracle Access Manager (provides more comprehensive user access capabilities).

Most, if not all, SSO technologies use a session cookie to hold encrypted data passed to each application. The SSO infrastructure has the responsibility to validate these cookies and, possibly, update this information. The user is directed to log on only if the cookie is not present or has become invalid. These session cookies are restricted to a single browser session and are never written to a file.

Another facet of SSO is how these technologies redirect a user's Web browser to various servlets. The SSO implementation determines when and where these redirects occur and what the final screen shown to the user is.

Most SSO implementations are performed in an application's infrastructure and not in the application logic itself. Applications that leverage infrastructure-managed authentication (such as deploying specifying "Basic" or "Form" authentication) typically have little or no code changes when adapted to work in an SSO environment.

What Do I Need for Oracle Single Sign-On?

The nexus of an Oracle Single Sign-On system is the Oracle Identity Management Infrastructure installation. This consists of the following components:

- An Oracle Internet Directory (OID) LDAP server, used to store user, role, security, and other information. OID uses an Oracle database as the back-end storage of this information.
- An Oracle Single Sign-On servlet, used to authenticate the user and create the OSSO session cookie. This servlet is deployed within the infrastructure Oracle Application Server (OAS).
- The Delegated Administration Services (DAS) application, used to administer users and group information. This information may also be loaded or modified using standard LDAP Data Interchange Format (LDIF) scripts.
- Additional administrative scripts for configuring the OSSO system and registering HTTP servers.

Additional OAS servers will be needed to deploy the business applications leveraging the OSSO technology.

Can Oracle Single Sign-On Work with Other SSO Implementations?

Yes, OSSO has the ability to interoperate with many other SSO implementations, but some restrictions exist.

Oracle Single Sign-on Terms and Definitions

Authentication

Authentication is the process of establishing a user's identity. There are many types of authentication. The most common authentication process involves a user ID and password.

Dynamically Protected URLs

A "Dynamically Protected URL" is a URL whose implementing application is aware of the OSSO environment. The application may allow a user limited access when the user has not been authenticated. Applications that implement dynamic OSSO protection typically display a "Login" link to provide user authentication and gain greater access to the application's resources.

Identity Management Infrastructure

The Identity Management Infrastructure is the collection of product and services that provide Oracle Single Sign-on functionality. This includes the Oracle Internet Directory, an Oracle HTTP server, and the Oracle Single Sign-On services. The Oracle Application Server deployed with these components is typically referred to as the "Infrastructure" instance.

MOD_OSSO

mod_osso is an Apache Web Server module that an Oracle HTTP Server uses to function as a partner application within an Oracle Single Sign-On environment. The Oracle HTTP Server is based on the Apache HTTP Server.

Oracle Internet Directory

Oracle Internet Directory (OID) is an LDAP-compliant directory service. It contains user ids, passwords, group membership, privileges, and other attributes for users who are authenticated using Oracle Single Sign-On.

Partner Application

A partner application is an application that delegates authentication to the Oracle Identity Management Infrastructure. One such partner application is the Oracle HTTP Server (OHS) supplied with the Oracle Application Server. OHS uses the MOD_OSSO module to configure this functionality.

All partner applications must be registered with the Oracle Single Sign-On server. An output product of this registration is a configuration file the partner application uses to verify a user has been previously authenticated.

Realm

A Realm is a collection of users and groups (roles) managed by a single password policy. This policy controls what may be used for authentication (for example, passwords, X.509 certificates, and biometric devices). A Realm also contains an authorization policy used for controlling access to applications or resources used by one or more applications.

A single OID can contain multiple Realms. This feature can consolidate security for retailers with multiple banners or to consolidate security for multiple development and test environments.

Statically Protected URLs

A URL is considered to be "Statically Protected" when an Oracle HTTP server is configured to limit access to this URL to only SSO authenticated users. Any attempt to access a "Statically Protected URL" results in the display of a login page or an error page to the user.

Servlets, static HTML pages, and JSP pages may be statically protected.

What Single Sign-On Is Not

Single Sign-On is not a user ID/password mapping technology.

However, some applications can store and retrieve user IDs and passwords for non-SSO applications within an OID LDAP server. An example of this is the Oracle Forms Web Application framework, which maps OSSO user IDs to a database login on a per-application basis.

How Oracle Single Sign-On Works

Oracle Single Sign-On involves a couple of different components. These are:

- The Oracle Single Sign-On (OSSO) servlet, which is responsible for the back-end authentication of the user.
- The Oracle Internet Directory LDAP server, which stores user IDs, passwords, and group (role) membership.

- The Oracle HTTP Server associated with the web application, which verifies and controls browser redirection to the OSSO servlet.
- If the web application implements dynamic protection, then the web application itself is involved with the OSSO system.

Statically Protected URLs

When an unauthenticated user accesses a statically protected URL, the following occurs:

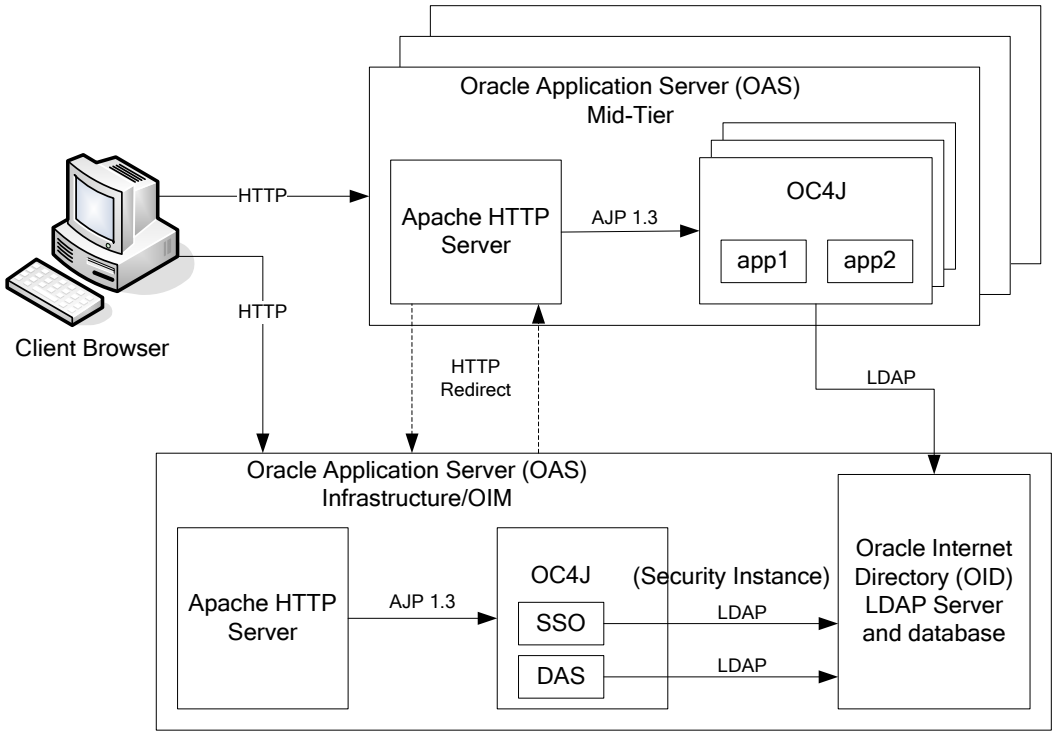
1. The Oracle HTTP server recognizes the user has not been authenticated and redirects the browser to the Oracle Single Sign-On servlet.
2. The OSSO servlet determines the user must authenticate, and displays the OSSO login page.
3. The user must sign in via a valid user ID and password. If the OSSO servlet has been configured to support multiple Realms, a valid realm must also be entered. The user ID, password, and realm information is validated against the Oracle Internet Directory LDAP server.
4. The OSSO servlet creates and sends the user's browser an OSSO session cookie. This cookie is never persisted to disk and is specific only to the current browser session. This cookie contains the user's authenticated identity. It does NOT contain the user's password.
5. The OSSO servlet redirects the user back to the Oracle HTTP Server, along with OSSO specific information.
6. The Oracle HTTP Server decodes the OSSO information, stores it with the user's session, and allows the user access to the original URL.

Dynamically Protected URLs

When an unauthenticated user accesses a dynamically protected URL, the following occurs:

1. The Oracle HTTP server recognizes the user has not been authenticated, but allows the user to access the URL.
2. The application determines the user must be authenticated and sends the Oracle HTTP server a specific status to begin the authentication process.
3. The Oracle HTTP Server redirects the user's browser session to the OSSO Servlet.
4. The OSSO servlet determines the user must authenticate, and displays the OSSO login page.
5. The user must sign in via a valid user ID and password. If the OSSO servlet has been configured to support multiple Realms, a valid realm must also be entered. The user ID, password, and realm information is validated against the Oracle Internet Directory LDAP server.
6. The OSSO servlet creates and sends the user's browser an OSSO session cookie. This cookie is never persisted to disk and is specific only to the current browser session. This cookie contains the user's authenticated identity. It does NOT contain the user's password.
7. The OSSO servlet redirects the user back to the Oracle HTTP Server, along with OSSO specific information.
8. The Oracle HTTP Server decodes the OSSO information, stores it with the user's session, and allows the user access to the original URL.

Single Sign-on Topology



Oracle Single Sign-On With WebStart Applications

This section describes a method devised by Oracle Retail for using Oracle Single Sign-On with Swing based clients that use Java WebStart technology. An EAR file, JnlpLaunchServlet.ear, has been created to aid an application in generating a temporary password, combined with OSSO, to create a secure launch of a Swing-based or 'fat' client.

How It All Works

The system works as follows:

1. The JnlpLaunch ear is deployed. Included in this EAR file is the JnlpLaunchServlet WAR file. The JnlpLaunch servlet is accessed to obtain the actual JNLP file used to launch the client application.
2. The JnlpLaunch servlet ensures that the Oracle Single Sign-On server has authenticated the user. If not, JnlpLaunch servlet forces the user to log in to the OSSO server.
3. Once the user's authenticated identity has been obtained, the JnlpLaunch servlet generates a temporary password based on the identity and other information.
4. The request is then internally forwarded to the JnlpGen servlet, which accesses a template file to create the final JNLP information. This template file contains tokens for the user name and password, which are replaced by the real user name and the temporary password. The JnlpLaunch servlet must be invoked with the name of the JNLP template file to use through the **template** parameter. A sample URL for the JnlpLaunch EAR is

http://www.jnlplaunch.mycompany.com/jnlplaunch/launch?template=mytemplate.vm

5. Finally, the JNLP information is returned to the browser, which kicks off its local copy of Java WebStart to download jars (if needed) and start the client application. Because the user name and temporary password are found within JNLP information, the client can now attempt to log into its server with these parameters.

> **Note:** The temporary password used is truly temporary. The server has a configuration option to determine how many seconds the password is valid. It will be up to the application administrator to determine this value and one can expect that an initial download of the application will be longer than this value. As such, the initial invocation will fail because the password has timed out, but subsequent invocations should succeed.

The JnlpLaunch EAR has the capability to be self-contained for an application. One can copy all of an application's signed jars and JNLP template files into this EAR either before deploying the EAR or post-deployment.

The JnlpLaunch WAR

JnlpLaunch WAR contains three servlets:

- JnlpLaunchServlet — used to validate a user name and creates a temporary password based on certain runtime and static configuration information. It then forwards its requests to the JnlpGenServlet.
- JnlpGenServlet — leverages the Apache Velocity project to create the JNLP information returned.
- UserInfoServlet — displays user request information and can be used to determine if the WAR has been properly installed.

Configuration For The JnlpLaunch Servlet

The WAR file assumes that in its classpath it has access to a configuration file called JnlpLaunch.properties. There are certain entries in this file that must be shared between the JnlpLaunchServlet and an application validating the passwords the JnlpLaunchServlet creates.

Installation Overview

Installing Oracle Single Sign-On consists of installing the following components:

1. Installing the Oracle Internet Directory (OID) LDAP server and the Infrastructure Oracle Application Server (OAS). These are typically performed using a single session of the Oracle Universal Installer and are performed at the same time. OID requires an Oracle relational database and if one is not available, the installer will also install this as well.

   The Infrastructure OAS includes the Delegated Administration Services (DAS) application as well as the OSSO servlet. The DAS application can be used for user and realm management within OID.
2. Installing additional OAS 10.1.2 midtier instances for the Oracle Retail applications, such as RMS, that are based on Oracle Forms technologies. These instances must be registered with the Infrastructure OAS installed in step 1.
3. Installing additional application servers to deploy other Oracle Retail applications and performing application specific initialization and deployment activities.

For more information about setting up Oracle Retail Store Inventory Management to use Oracle Retail Workspace, see "Setting up Oracle Retail Store Inventory Management to use Oracle Single Sign-On".

For more information about integrating Oracle Retail Store Inventory Management with Oracle Retail Workspace, see "Integration with Oracle Retail Workspace" in the *Oracle Retail Store Inventory Management Operations Guide*.

Infrastructure Installation and Configuration

The Infrastructure installation for OSSO is dependent on the environment and requirements for its use. Deploying an Infrastructure OAS to be used in a test environment does not have the same availability requirements as for a production environment. Similarly, the Oracle Internet Directory (OID) LDAP server can be deployed in a variety of different configurations. See the *Oracle Application Server Installation Guide* and the *Oracle Internet Directory Installation Guide* for more details.

OID User Data

Oracle Internet Directory is an LDAP v3 compliant directory server. It provides standards-based user definitions out of the box.

The current version of Oracle Single Sign-On only supports OID as its user storage facility. Customers with existing corporate LDAP implementations may need to synchronize user information between their existing LDAP directory servers and OID. OID supports standard LDIF file formats and provides a JNDI compliant set of Java classes as well. Moreover, OID provides additional synchronization and replication facilities to integrate with other corporate LDAP implementations.

Each user ID stored in OID has a specific record containing user specific information. For role-based access, groups of users can be defined and managed within OID. Applications can thus grant access based on group (role) membership saving administration time and providing a more secure implementation.

OID with Multiple Realms

OID and OSSO can be configured to support multiple user Realms. Each realm is independent from each other and contains its own set of user IDs. As such, creating a new realm is an alternative to installing multiple OID and Infrastructure instances. Hence, a single Infrastructure OAS can be used to support many development and test environments by defining one realm for each environment.

Realms may also be used to support multiple groups of external users, such as those from partner companies. For more information on Realms, see the *Oracle Internet Directory Administrators Guide*.

User Management

User Management consists of displaying, creating, updating or removing user information. There are two basic methods of performing user management: LDIF scripts and the Delegate Administration Services (DAS) application.

OID DAS

The DAS application is a web-based application designed for both administrators and users. A user may update their password, change their telephone number of record, or modify other user information. Users may search for other users based on partial strings of the user's name or ID. An administrator may create new users, unlock passwords, or delete users.

The DAS application is fully customizable. Administrators may define what user attributes are required, optional or even prompted for when a new user is created.

Furthermore, the DAS application is secure. Administrators may also define what user attributes are displayed to other users. Administration is based on permission grants, so different users may have different capabilities for user management based on their roles within their organization.

LDIF Scripts

Script based user management can be used to synchronize data between multiple LDAP servers. The standard format for these scripts is the LDAP Data Interchange Format (LDIF). OID supports LDIF script for importing and exporting user information. LDIF scripts may also be used for bulk user load operations.

User Data Synchronization

The user store for Oracle Single Sign-On resides within the Oracle Internet Directory (OID) LDAP server. Oracle Retail applications may require additional information attached to a user name for application-specific purposes and may be stored in an application-specific database. Currently, there are no Oracle Retail tools for synchronizing changes in OID stored information with application-specific user stores. Implementers should plan appropriate time and resources for this process. Oracle Retail strongly suggests that you configure any Oracle Retail application using an LDAP for its user store to point to the same OID server used with Oracle Single Sign-On.

Setting up Oracle Retail Store Inventory Management to use Oracle Single Sign-On

Before installing Oracle Retail Store Inventory Management, you must install the infrastructure server or Oracle Identity Management (OIM) server.

SIM leverages the JnlpLaunch servlet to provide an Oracle Single Sign-On (OSSO) launch of the SIM client. This servlet uses the authenticated OSSO user ID in creating a Java Network Launch Protocol (JNLP) message to the client browser. It also generates a temporary password that, along with the SSO user name, is used by the client to log into the SIM server.

The temporary password generated by the JnlpLaunch servlet is valid only for a configured amount of time. The initial download of the SIM client may take longer than the timeframe in which the temporary password is valid. If so, the temporary password becomes invalid and a user will be presented with a login dialog. However, subsequent launches of the client will not need to download the entire application and should be much faster.

The JnlpLaunch Servlet is a dynamically protected application. That means that no entries are needed in the HTTP Server's mod_osso.conf file to specifically protect the SIM client launch. However the HTTP Server must still be configured with OSSO, registered with the OSSO server, and reference the osso.conf file that was created during the registration process. For more details on this process, see the Oracle Single Sign-On documentation.

> **Note:** SIM has no dependency on Oracle Retail Workspace. Oracle Single Sign-On can be enabled for SIM regardless of whether Oracle Retail Workspace is installed. Similarly, the JnlpLaunch servlet may be configured for use in a non-OSSO environment.

JnlpLaunch.properties in SIM

The JnlpLaunch Servlet uses a template file to generate the JNLP message sent to a user's browser to download or launch the SIM client. When used in an OSSO environment, the JnlpLaunch servlet will generate a temporary password based on the OSSO user name and properties found in the JnlpLaunch.properties file. While some properties are specific only to validation or generation, for example, the length of time the password is valid, other properties must be the same between the JnlpLaunch servlet and the SIM login service. Both the JnlpLaunch servlet and the SIM application reference a single copy of the JnlpLaunch.properties file.

SIM is deployed as two EAR files, sim-client.ear and sim13.ear. The sim-client.ear file contains a sim-client.war file, which contains the classes and deployment descriptors for the JnlpLaunch servlet.

Both sim13.ear and sim-client.ear define a shared library reference to `<SIM_OC4J_INSTANCE>/sim-home/files/prod/config`. The shared library is an Oracle Application Server feature that enables an application to include the contents of directories outside the EAR file in the application's classpath.

Because both sim13.ear and sim-client.ear have this directory in their classpath, this is where the JnlpLaunch.properties file is kept. This way the application is guaranteed that the JnlpLaunch servlet (which generates the temporary password) and the SIM server application (which validates the shared password) both reference the same JnlpLaunch.properties values.

2

Setup and Configuration

Setting up Security

Overview

SIM allows users and menu items to be assigned privileges that control available menu options.

- Users are given privileges in Lightweight Directory Access Protocol (LDAP)
- Handheld menu items are given privileges in a Java class
- PC menus are given privileges in navigation.xml

SIM security is supported only for users defined in LDAP. When a user has been assigned a role, in LDAP, then every menu item with that privilege's number is accessible. The exceptions to this are:

- Users can be defined as super users who have access to everything
- Buttons assigned a task_item_permission of 0 in navigation.xml are accessible to everyone

Handheld menu privileges are assigned fixed numbers hardcoded in Authorization.java. To gain access to a handheld menu a user must have a privilege corresponding to one of these numbers. Any number that is a power of two can be given to any button in navigation.xml and user role in LDAP. The maximum value allowed is 264, which is 18446744073709551616.

How SIM Associates Menu, Menu Items and Windows

Menus and buttons are defined in navigation.xml. The SIM navigation framework parses and uses this xml to display buttons. If the privilege assigned to the button does not exist for the store and user, the button is not shown when the navigation menu is displayed.

SIM Privilege Definitions

The privileges in the following table are used in SIM. These values are used in the following places:

1. Authorization.java to control access to handheld screens.
2. navigation.xml to control access to PC screens.
3. LDAP role records to assign which privileges a given role has.
4. LDAP store records to determine which privileges a store may have.

Privilege Description	Value
Create/View Stock Count (my store)	1
Create/View Stock Count (all stores)	2
Authorize Count	4
Item Lookup	8

Privilege Description	Value
Transfer Receive	16
Transfer Create/Save	32
Supplier Lookup	64
DSD	128
Return Stock	256
Warehouse Delivery	512
Container Lookup	1024
Inventory Adjustment	2048
Pricing	4096
View/Perform Stock Count	8192
Store Admin/Configuration	16384
Transfer Requests	32768
Item Requests	65536
Sequencing	131072
Pick Lists	262144
Store Orders	524288
Item Tickets	1048576
Price Change	2097152
Adjust Delivery	4194304

LDAP User Privilege Configuration

This section describes LDAP user privileges, the privileges hierarchy, and the way in which menu access privileges are granted to SIM users.

Super Users

Privileges do not restrict menu access to super users. The rest of this document applies to non-super users only.

The Privilege Hierarchy

Each menu item can be given a privilege restriction. When a menu item has a privilege, the button displays when a store or user has the privilege for it.

As an example of store level restrictions, take a menu in which restrictions have been placed on the Transfer Receive, Return Stock, Warehouse Delivery and Pricing items. When this menu is displayed by a store that has privileges only for the first three items, then the Pricing button will not appear.

Users may also be restricted and these restrictions are applied after store level restrictions are applied. For example, if a user has privileges for Transfer Receive, Return Stock and Container Lookup, looking at the store described previously, this user sees only the Transfer Receive and Return Stock buttons. The Container Lookup button is not shown to the user because it is not accessible to the store. If the store had privileges to all menu items, then this user would see the three buttons for which privileges were granted.

Granting Privileges to Users

A *role* is a group of privileges. Privileges are placed in roles and roles are assigned to users, that is, roles associate privileges with users. Roles are created as LDAP rsimRoles entries and contain one or more privilege attributes. Users, created as LDAP rsimUsers entries, are assigned a role in the entry's userRole attribute. Note that role privileges cannot override store privileges. For example, if privilege 512 is not available to a store, users logged into that store will not be able to perform function 512 even if their role specifically allows it.

Once an LDAP user is correctly set up for a store that is present in your SIM database, you will be able to log in to the SIM client.

Algorithm that Determines if Item is Displayed

A bitwise `AND` operation determines if a menu item is to be displayed. The operation is done in the Employee.hasAnyRoleFromRoles (…) method.

This algorithm works by adding together all privileges for the role, then `AND`ing with the restriction on a particular button. For example, for a menu item with a restriction of 16 and a role with privileges 1, 2, 8 and 32, the sum of the privileges is 43. Performing an `AND` on 43 and 16 yields 0, indicating the role lacks the 16 privilege. For a menu item of 8, the result of the `AND` is non-zero, indicating the privilege exists.

Privilege Examples

Example of a Store Definition

The following is the LDAP definition of store 1000000031. This store is assigned privileges 1 through 8192.

```
dn: rsimStoreId=1000000031
privilege: 1
privilege: 2
privilege: 4
privilege: 8
privilege: 16
privilege: 32
privilege: 64
privilege: 128
privilege: 256
privilege: 512
privilege: 1024
privilege: 2048
privilege: 4096
privilege: 8192
rsimStoreId: 1000000031
```

Neither the System Admin nor Store Admin buttons are displayed because the store lacks privileges 16384 and 262144. Though a user may be eligible to see the Store Admin button (privilege 16384), this privilege does not exist for the store so the button is not shown.

Example of a Role Definition

The following is an LDAP definition of a role named **Store Manager**. It has privileges 8, 16 and 256. A user assigned this role is restricted to only viewing menu items that have privilege values of 8, 16, 256 and 0. In the following menu, users with role Store Manager will not see the System Admin button because its privilege, 262144, is not a privilege contained in the Store Manager role.

```
dn: rsimRoleName=Store Manager
privilege: 8
```

```
privilege: 16
privilege: 256
isSuperUser: FALSE
isStoreSuperUser: FALSE
rsimRoleName: Store Manager
```

Example of a User Definition

Below is an LDAP definition of a user named 9903StoreManager. This user has been assigned the Store Manager role.

```
dn: uid=9903StoreManager,cn=Users,dc=SystemTest,dc=oracle,dc=com
employmentStatus: 0
ssn: 123456789
userPassword:: e1NIQX1TcVhHeWNvZXBXZGM3ejh1SFg2akhoalRLYkU9
shortName: 9903StoreManager
firstName: 9903StoreManager
lastName: 9903StoreManager
objectClass: rsimUser
uid: 9903StoreManager
userStore: rsimStoreId=9903,cn=rsimStores,cn=RSIM,dc=SystemTest,dc=oracle,dc=com
email: 9903StoreManager@retek.com
preferredCountry: UK
preferredLanguage: En
userRole: rsimRoleName=Store
Manager,cn=rsimRoles,cn=RSIM,dc=SystemTest,dc=oracle,dc=com
homeStore: rsimStoreId=9903,cn=rsimStores,cn=RSIM,dc=SystemTest,dc=oracle,dc=com
middleName: xxx
```

Example of Assigning Restrictions to Menu Buttons

In the following excerpt from a navigation.xml file, the Main Menu button is available to all users because the task_item_permission is set to **0**. The remaining menu items will only be available to stores and users granted privileges 8 and 1024.

```
    <task>
    <task_name>oracle.retail.sim.shared.swing.login.LookupMenuScreen</task_name>
        <default_task_item>Default</default_task_item>
        <task_item>
            <task_item_name>Main Menu</task_item_name>
            <task_item_permission>0</task_item_permission>
            <task_item_command>BACK</task_item_command>
            <task_item_duplicate>false</task_item_duplicate>
        </task_item>
        <task_item>
            <task_item_name>Item Lookup</task_item_name>
            <task_item_permission>8</task_item_permission>

    <task_item_command>oracle.retail.sim.shared.swing.item.ItemLookupScreen</task_
item_command>
            <task_item_duplicate>false</task_item_duplicate>
        </task_item>
        <task_item>
            <task_item_name>Supplier Lookup</task_item_name>
            <task_item_permission>64</task_item_permission>

    <task_item_command>oracle.retail.sim.shared.swing.supplier.SupplierLookupScree
n</task_item_command>
            <task_item_duplicate>false</task_item_duplicate>
        </task_item>
        <task_item>
            <task_item_name>Container Lookup</task_item_name>
            <task_item_permission>1024</task_item_permission>
```

```
    <task_item_command>oracle.retail.sim.shared.swing.carton.CartonLookupScreen</t
ask_item_command>
            <task_item_duplicate>false</task_item_duplicate>
        </task_item>
    </task>
```

Setting up LDAP Data for SIM

Setup of LDAP

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.

> **Note:** If you are using Oracle Single Sign On with SIM, it is required to use Oracle Internet Directory as your LDAP implementation (see "Oracle Single Sign-on Overview").

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:

> **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_data_role_store_containers.ldif

  The containers for holding stores and roles. This file must be modified before it is imported into your LDAP system (see readme.txt).
- sim_data_roles.ldif

  Sample role data. This file must be modified before it is imported into your LDAP system (see readme.txt).
- sim_data_stores.ldif

  Sample store data. This file must be modified before it is imported into your LDAP system (see readme.txt).
- sim_data_users.ldif

  Sample user data. This file must be modified before it is imported into your LDAP system (see readme.txt).

Note: You can have more than one rsimStoreID by simply repeating the userStore line, but should only have one homeStore.

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

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 via a 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 below.

Setting the Time Zone

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 4, "System and Store Administration".

- REPORTING_TOOL_ADDRESS
- REPORTING_TOOL_REQUEST_URL
- REPORTING_TOOL_REQUEST_USERNAME
- REPORTING_TOOL_REQUEST_PASSWORD
- REPORT_FORMAT_DIRECTDELIVERY
- REPORT_FORMAT_ITEMDETAIL
- REPORT_FORMAT_ITEMREQUEST
- REPORT_FORMAT_ITEM_TICKET
- REPORT_FORMAT_PICKLIST
- REPORT_FORMAT_RETURN
- REPORT_FORMAT_SHELF_LABEL
- REPORT_FORMAT_STOCKCOUNT
- REPORT_FORMAT_STOCKCOUNTALLLOC
- REPORT_FORMAT_STOCKRECOUNT
- REPORT_FORMAT_STOCKCOUNT_NOF
- REPORT_FORMAT_STOREORDER
- REPORT_FORMAT_TRANSFER
- REPORT_FORMAT_WAREHOUSEDELIVERY

Data Seeding

Overview

This section provides the details for seeding the data from Oracle Retail Merchandising System (RMS) to the SIM database. The data seeding scripts perform the following:

- Move the data from the RMS database to the SIM database.
- Performs basic tests of the movement of data from source RMS schema to target SIM schema, and provides a detailed log.

The data seeding scripts were originally written as java programs. They have now been moved to sql loader scripts. The scripts have similar logic to the java programs.

Executing Script

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

- It is highly recommended to backup the database before executing the data seeding scripts.
- Performance can be an issue when a large amount of data needs to be migrated in a limited timeframe. Retailers could look into testing the scripts in their pre-production or test environment with a limited amount of data.
- If there are any custom modifications to the database schema, then those need to be handled by the custom-modification or retailer team.
- Any bad data, due to null values or missing constraints, identified during seeding should be manually moved to the new SIM schema.

> **Note:** The SIDs for RMS and SIM databases should exist in the machine where these data seeding scripts are run.

1. In the `dataseeding.sh` file, update the following:

- SIM database name (SIM_DB)
- SIM database user (SIM_USER)
- SIM database password (SIM_PWD)
- RMS database name (RMS_DB)
- RMS database user (RMS_USER)
- RMS database password (SIM_PWD)

2. The STORES_LIST should also be specified in `dataseeding.sh`. If the STORES_LIST is set to **ALL**, dataseeding would run for all the stores. If the process needs to run only for specific stores, then the stores list needs to be given in a comma separated format.
3. The SIM_COMMIT_BLOCK indicates the number of records that are read by the SQL loader. This also needs to be set before executing the script.
4. The DIRECT_SQL_LOAD should also be specified in `dataseeding.sh`. For improved performance, set it to **DIRECT_SQL_LOAD=TRUE** for faster loading.
5. At the command prompt, run `dataseeding.sh` to seed the data from RMS schema to SIM schema.
6. After the completion of the data seeding, verify the dataseeding.log file for any errors.

Following are the main script files and their usage description:

File Names	Description
dataseeding.sh	This is the main shell script file, which controls the seeding of the data from RMS to SIM schema. The following are the steps of execution within this script: 1. Disable all the Foreign Key constraints in SIM database by executing `DisableForeignKeys.sh` 2. Create the temp tables required for a few message families like Item, Store, Warehouse, Supplier and System Config. 3. Create the packages required for executing the data seeding process. Certain message families use the packages for inserting data into the SIM schema. The sql loader loads data into the temp tables. After data is loaded in the temp tables, the stored procedures insert data into the required SIM tables. 4. While invoking each child script, get the Process ID of each process. 5. Keep pinging for the completion of each child process. 6. Once all the child process execution is completed, enable all the Foreign Key constraints by executing `EnableForeignKeys.sh` 7. Execute the `Test.DataSeeding.sh` file, which does the basic check of the seeding of the total number of records that are moved for each message family.
DisableForeignKeys.sh	This shell script file executes the `DisableForeignKeys.sql` file at the SQLPLUS prompt to disable all the Foreign Key constraints on SIM database. This is done to disable any checks at the load of the data to SIM schema.
DisableForeignKeys.sql	This SQL file spools the scripts to disable all the Foreign Key constraints to temporary file `DisableFks.sql`. This temp file is then executed to actually disable the Foreign Key constraints.
CreateTempTables.sh	This shell script creates the temporary tables for inserting data into actual tables at the SIM `end.sql` loader is used to insert tables from RMS database to the temporary tables. The temp tables are created for Item, Supplier, Config, Store and Warehouse message families.

File Names	Description
CreateTempTables.sql	This creates the following temp tables in SIM database: ▪ Store_temp ▪ warehouse_temp ▪ rk_config_temp ▪ item_master_temp ▪ supplier_temp
CreatePackage.sh	This shell script is used to create the stored procedures required to insert data into the actual SIM tables from the temporary tables. The stored procedures are used to insert data for Item, Supplier, Config, Store and Warehouse.
CreatePackage.sql	This creates the four packages associated with loading of data into the actual SIM tables for Item, Supplier, Config, Store and Warehouse message families.
DropTempTables.sh	This shell script, after the successful execution of data seeding processes, drops all the temp tables created for the sole purpose of dataseeding.
DropTempTables.sql	This sql script drops the temp tables.
DropPackage.sh	This shell script, after the successful execution of dataseeding processes, drops the stored procedures that were created for dataseeding.
EnableForeignKeys.sh	This shell script file executes the `EnableForeignKeys.sql` file at the SQLPLUS prompt to enable all the Foreign Key constraints on SIM database.
EnableForeignKeys.sql	This SQL file spools the scripts to enable all the Foreign Key constraints to a temporary file `EnableFks.sql`. This temp file is then executed to actually enable the Foreign Key constraints.
Test.sh	This shell script checks the record count of each table from SIM to RMS. This executes the `Test.sql` file at the SQLPLUS prompt.
Test.sql	This file checks the count of records for each message family in the RMS database and compares it with corresponding records in the destination SIM tables. If the count matches, it is recorded as a success. Otherwise it is recorded as a failure.
DisableIndexes.sh	This shell script disables all the indexes before the loading of data into destination database. This is done to improve the performance.
DisableIndexes.sql	This sql script spools the list of indexes disabling scripts to a temp file `DisableIndex.sql` and then execute the spooled file to disable the indexes.

File Names	Description
RebuildIndexes.sh	This shell script enables all the indexes after the loading of data into destination database.
RebuildIndexes.sql	This sql script spools the list of indexes enabling scripts to a temp table `RebuildIndex.sql` and then execute the spooled file to enable the indexes.
Test.log	This log file contains information of the tables whose data have been successfully or unsuccessfully moved from RMS to SIM.

Apart from the above-mentioned files for data seeding, there are .sh and .ctl files for each table, which is migrated from RMS to SIM.

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.

Oracle Retail Store Inventory Management and SSO

For information on configuring and setting up Oracle Retail Store Inventory Management and SSO, go to "Setting up Oracle Retail Store Inventory Management to use Oracle Single Sign-On".

3

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 point of sale 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
- 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
- Email Alerts

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), Retail Warehouse Management System (RWMS), Oracle Retail Invoice Match (RIM), 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 process 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. The program enables all foundation item/location relevant data to be populated within the SIM database from the merchandising system.

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 dialogue 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 dialogue 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.

Inventory Management

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.
- System used inventory adjustment codes can be displayed to or hidden from the user.
- The system notifies the merchandise system of all inventory adjustments.

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. Note that retailers often configure the information on a database level to best fit their business needs. The abbreviations in the table correspond to the following names:

- A: Available
- O: Out
- U: Unavailable

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: The original pre-loaded codes listed in the following table are required for SIM to operate. Removing any of these reason codes will cause SIM to cease to function.

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

26	78	Unit and Amount Late Sales Increase SOH	O -> A
27	76	Unit Late Sales Increase SOH	O -> A

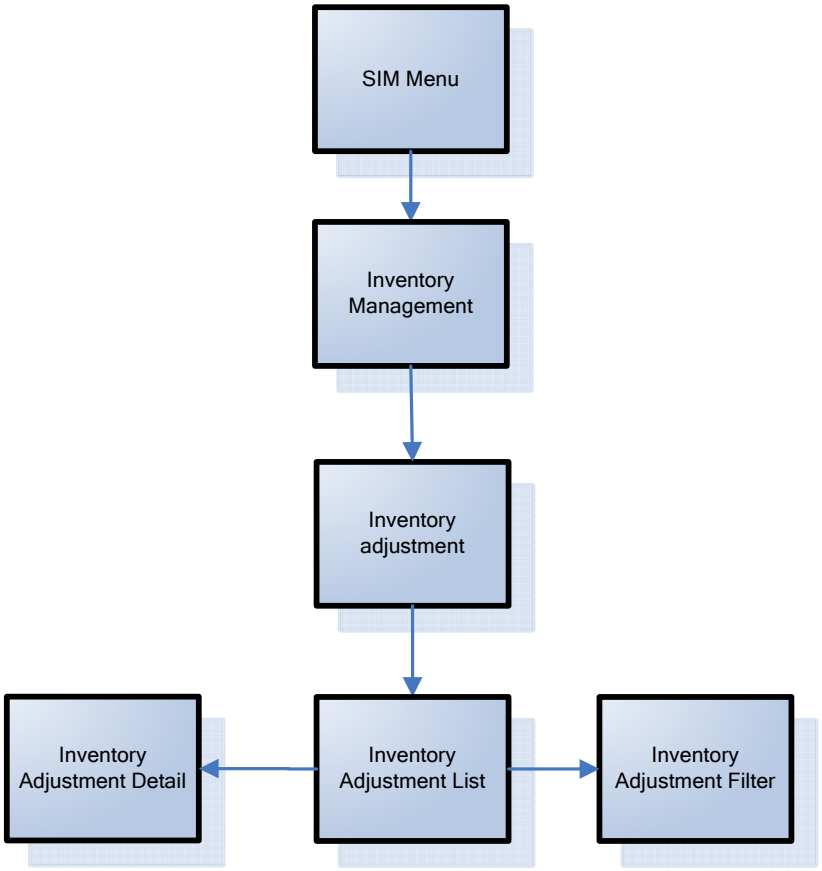
Updating Reason Codes

The reason codes in the table above are pre-loaded into the RK_INV_ADJ_REASON table. The Internal Code represents the identifier used by SIM to retrieve and use these reason codes during processing. The Internal Reason code cannot be altered for the reason code specified.

The External Code represents the identifier or code that is sent during any transaction that communicates with an external system. Messages published from SIM will contain this value as the reason code. To ensure an integrated system is working properly, the External Code must match the same reason code in the external system.

There is no UI to alter the External Code, so someone with database access will be required to handle these changes. The DBA will need to update the REASON_CODE column of the RK_INV_ADJ_REASON table with the appropriate values from the external system.

Business Process Flow – Inventory Adjustments PC



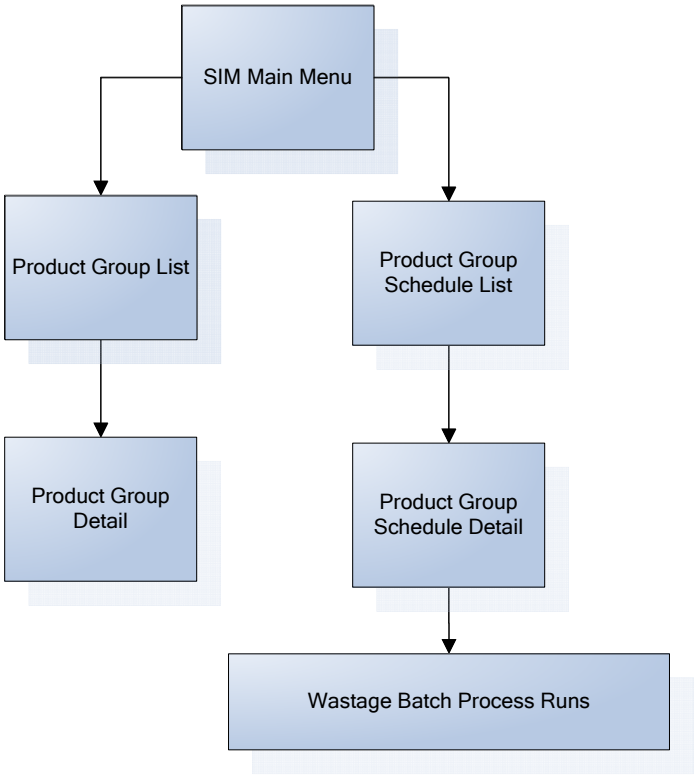
Wastage Functional Overview

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's wastage functionality provides users in stores with 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 RIB to the merchandising system.

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.

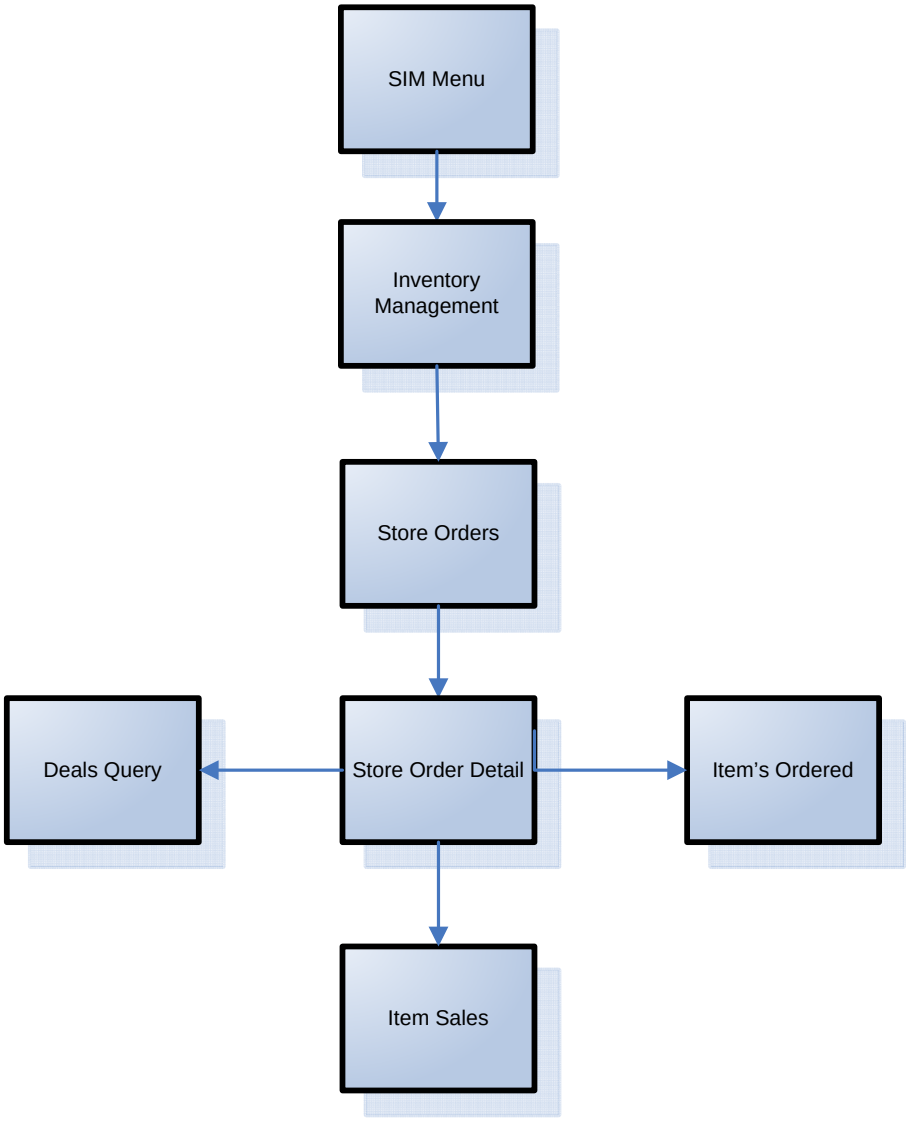
Store ordering functionality is very similar to item request functionality. Unlike item request functionality, which is only valid for items that are on the store order replenishment process, store order functionality is valid for all items, cannot be scheduled and is only available on the PC.

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.

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 (described above). 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.

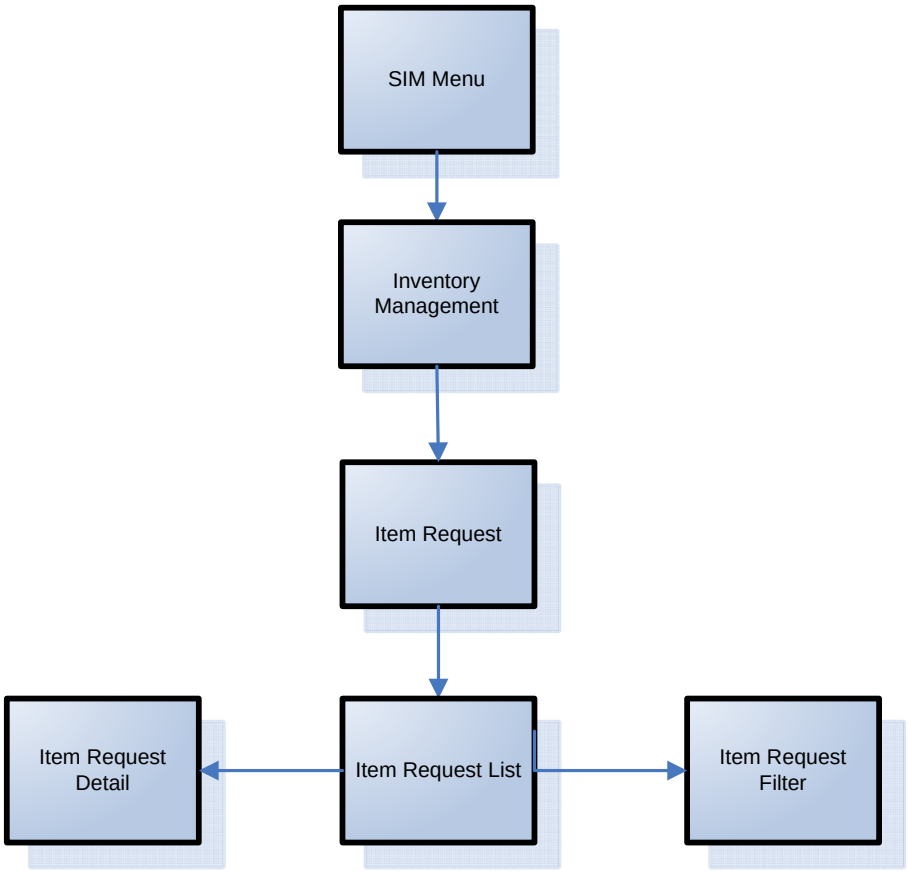
Any quantities requested for items that have a replenishment type other than 'Store Order' are added above and beyond the quantity that is normally sourced through the merchandising system on the item's next replenishment review date. However, if the requested delivery date falls prior to such an item's 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.

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 cyclic 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 list below 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.

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 RPM. RPM 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 RPM, they can use a flexible structure to control the retail prices via 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.

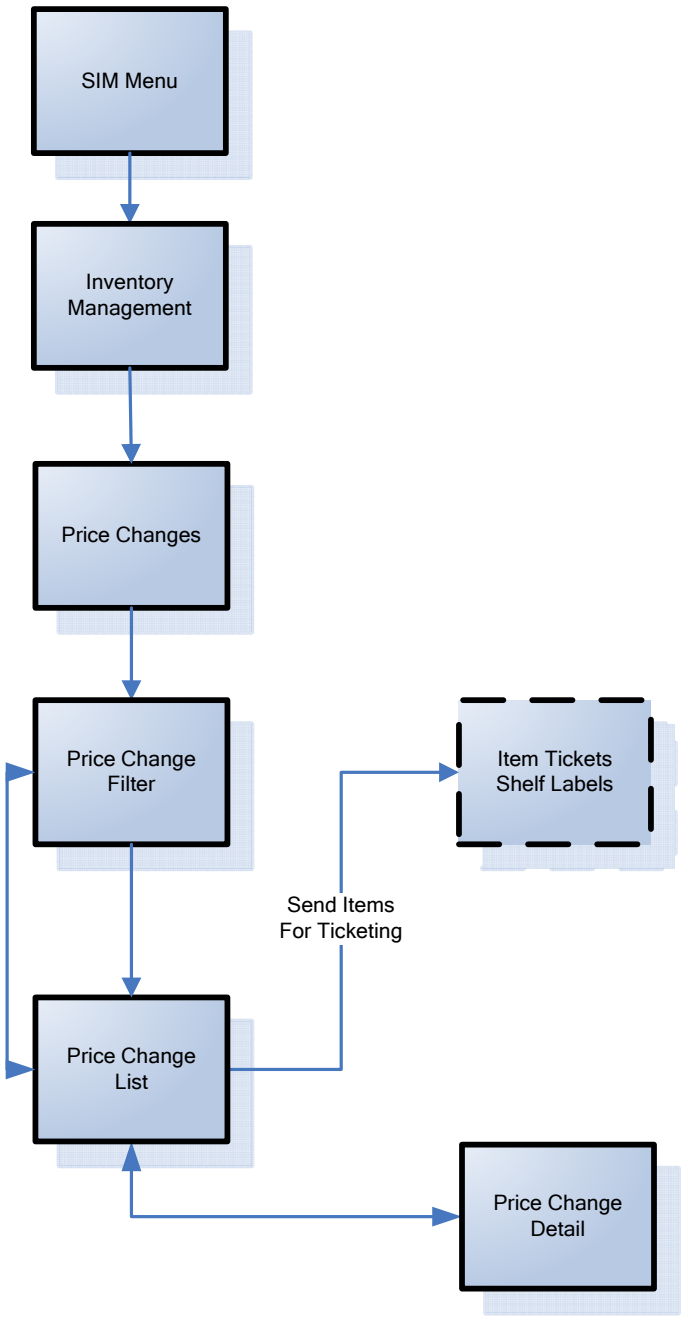
In RMS, an indicator at the item/location level determines whether SIM users can request changes to an item's retail price at a specific location. This indicator is editable and controls behavior going forward but not in the past.

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 RPM in real time. RPM checks for any conflicts and provides a response to SIM regarding the status of the request. If the request is accepted, RPM 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 RPM.
- 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 RPM as a new event request. It undergoes conflict checking in RPM. 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 RPM, but they have no control over them. SIM is unable to create new price change requests to be sent to RPM.
- Communication between SIM and RPM is handled by RSL (providing a near real-time connection between the applications). Normal operation of pricing assumes that RSL and RPM 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.

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 dialogues 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.

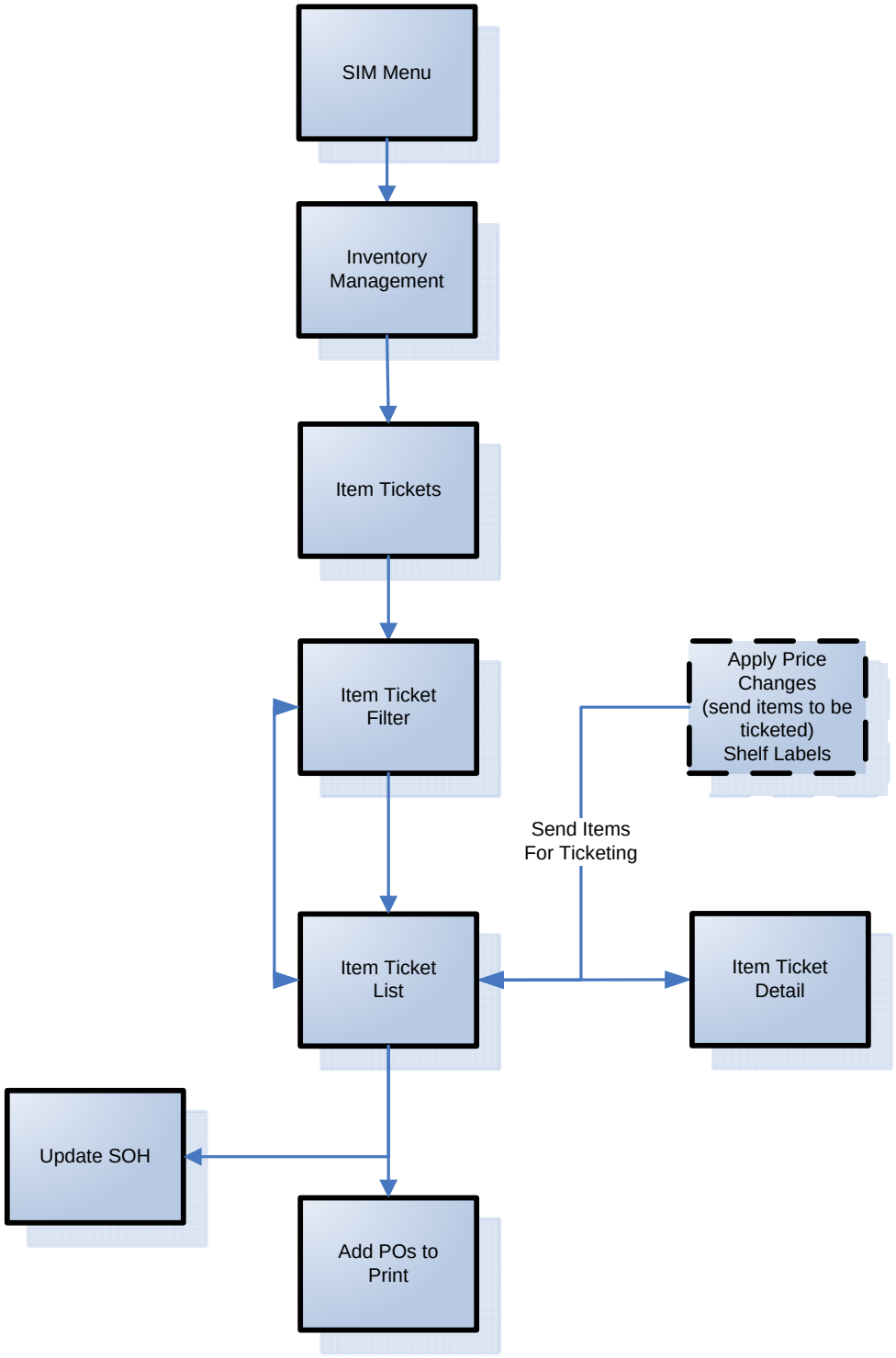
      Note: This override price does not generate a price change.

   b. Pricing dialogue —Users can also send tickets to the Item tickets dialogue 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 dialogue.
   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

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.

Business Process Flow – Ticketing PC



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).

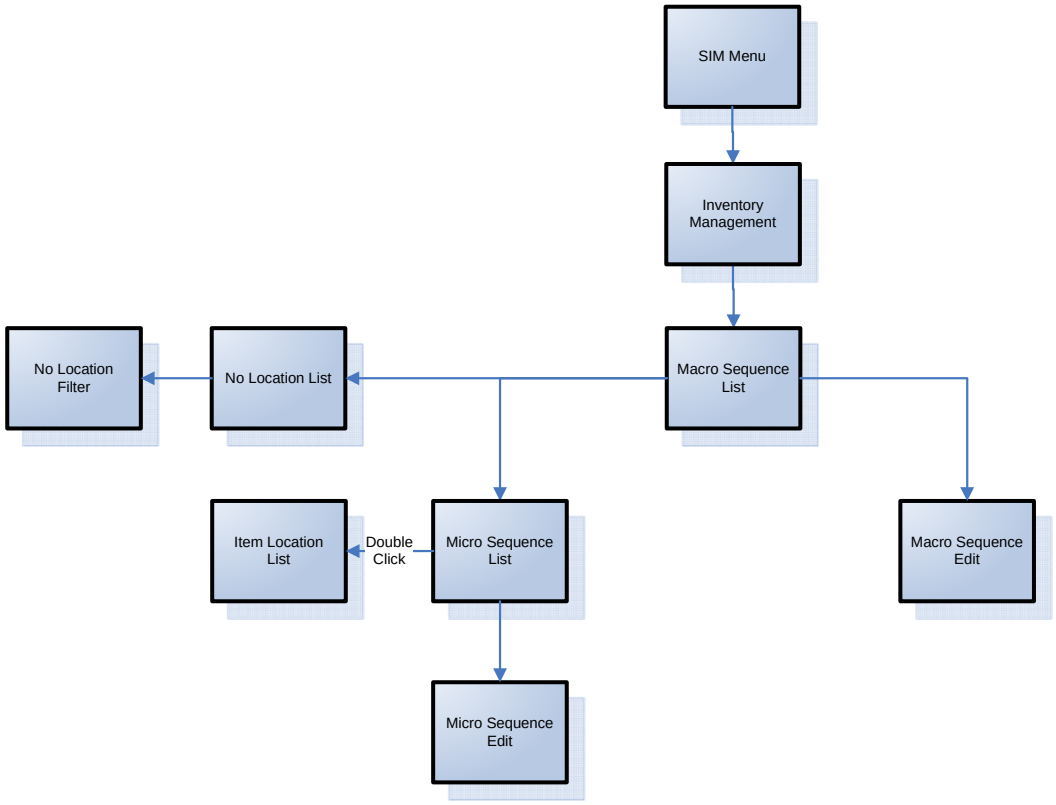
A micro sequence is the lowest (most granular) item location level.

The following table provides an example of sequencing:

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

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

Business Process Flow – Sequencing PC



Shelf Replenishment 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 and provides the container number 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, backroom, 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 backroom 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.

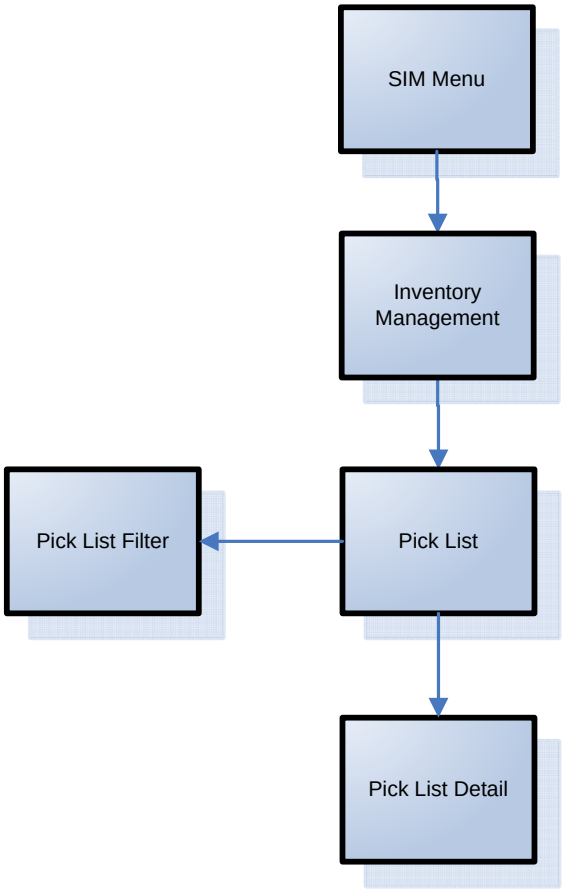
Pick lists can be created on the PC or the handheld and can be fulfilled on either device. If they are created on the PC, the user is able to action them on the handheld.

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 backroom, the system displays the items in item ID order.

Business Process Flow – Pick List 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 (wireless only)
- Unit only
- Unit and amount
- Unit and amount all departments
- Problem line
- Un-guided

> **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.

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. Wireless stock counts are divided so that each user of a handheld device performs a stock count according to a macro location (for example, vegetables, cereals, and so on).

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. Ad hoc stock counts are only performed on the handheld side of the system. 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.
- An ad hoc stock count can be saved and completed later; any existing ad hoc stock count can be retrieved and resumed.
- 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 the 'Inventory Adjustments' section, 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.

Unit only Stock Counts

Unit only stock count processing within SIM includes the following features:

- Individual items and item hierarchies are associated into a single unit product group for the purpose of scheduling a stock count.
- Unit 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.
- A stock count item list is generated at the store level via a batch process.
- 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.
- 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 the 'Inventory Adjustments' section, in this chapter.

Unit and Amount Stock Counts

Third-Party Unit and Amount Stock Counts

Retailers can perform their unit and amount stock counts using a third-party vendor. In such cases, the unit and amount 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 SOH value currently held in SIM. Within SIM, users can view all items in the stock count that are 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 thus should be considered discrepant.

Any items SIM does not recognize can be added through the Not On File dialogue.

Once the discrepant items have been reviewed and approved, SIM exports the stock count information to RMS. RMS updates its SOH positions to reflect the data held in SIM and makes applicable financial adjustments relevant to the unit and amount stock count.

It is also possible to have these counts automatically processed ensuring no store interference. For Sarbanes-Oxley Act requirements, this is the optimal way to process a unit and amount count. This automatic third-party process does not require any counting or approval from store personnel.

SIM Unit and Amount Stock Counts

The user can create one of the following types of unit and amount stock counts:

- Regular unit and amount stock count
  Item hierarchies are associated into a single unit and amount product group for the purpose of scheduling a stock count.
- All locations unit and amount stock count
  If the store uses sequencing, the user can associate all macro sequencing locations (which would include all items in the store) to a stock count.

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

- 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 value, percentage, or standard unit of measure by the item's class in the merchandise hierarchy.
- On the PC, the system requires the authorization of items that are discrepant (based on the value, percentage, or standard unit of measure thresholds).

Unit and Amount Stock Counts All Departments

- Unit and amount product groups can be scheduled for a stock count on a specified day.
- The stock count schedule for unit and amount stock counts is sent to the merchandising system anytime it is created/updated/deleted.
- One or more stores can be assigned to the scheduled stock count.
- A stock count item list is generated at the store level via a batch process (described below) that runs daily.
- 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.

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
- One or more stores can be assigned to the scheduled stock count.
- A problem line stock count item list is generated at the store level via a problem line batch process that is based on problem line parameters.
- 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.
- 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 the 'Inventory Adjustments' section, 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.

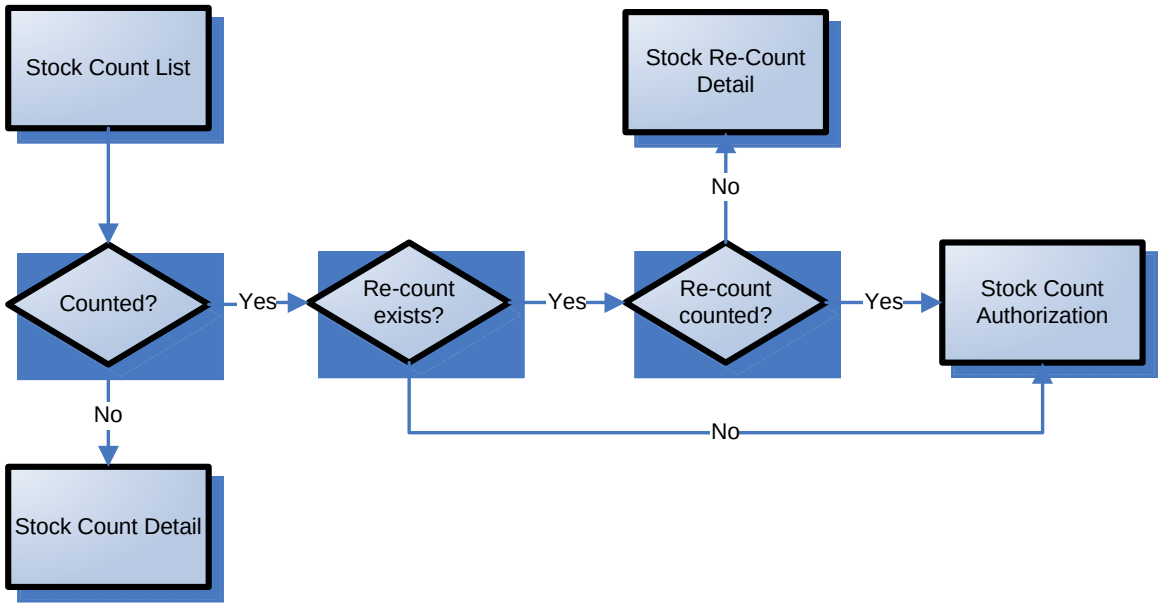
Unguided Stock Counts

When performing a scheduled stock count (unit or unit and amount), the user is able to scan items on the handheld without being prompted which item to scan. Multiple employees are able to scan items for the same stock count. Both these features are controlled through system options.

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.

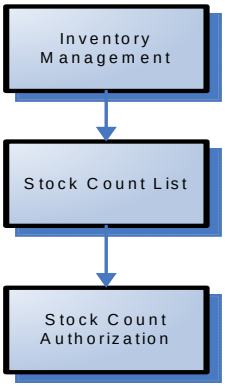
Guided stock counts will prompt the user for the next item in sequence. If sequencing is not set up for the items, the user will be prompted in item order.

Business flow Unit and Problem Line



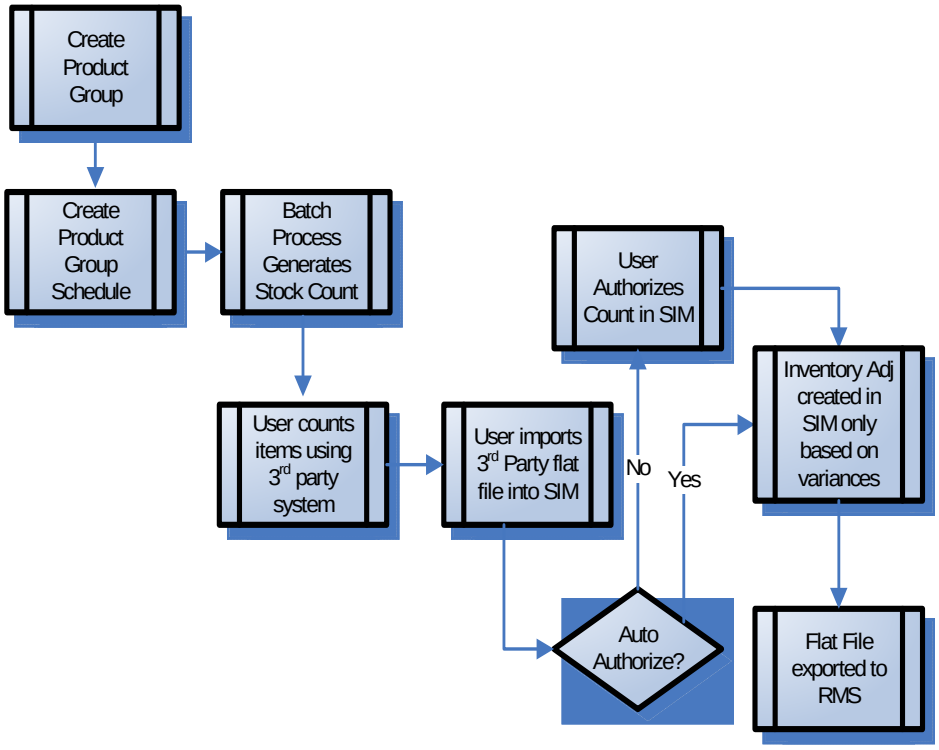
Business flow – Ad Hoc (PC only)

Note: Ad hoc count process is performed on the handheld. PC involves authorization process only.

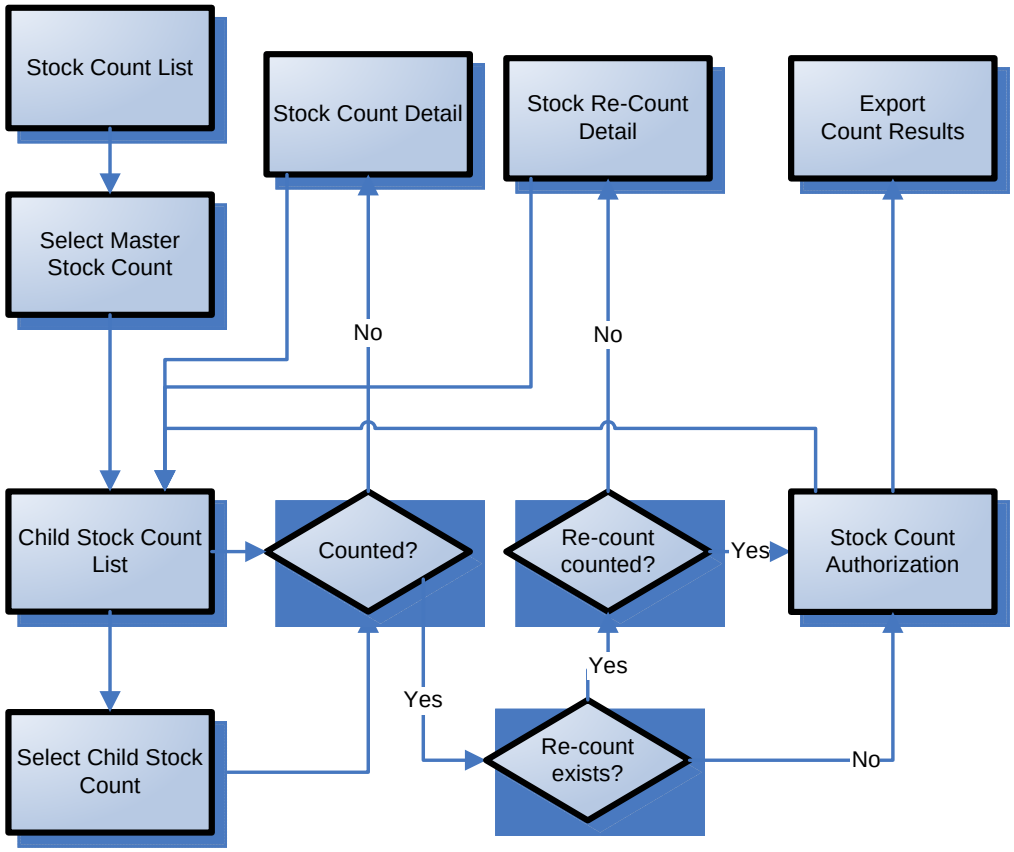


Business flow – Third Party

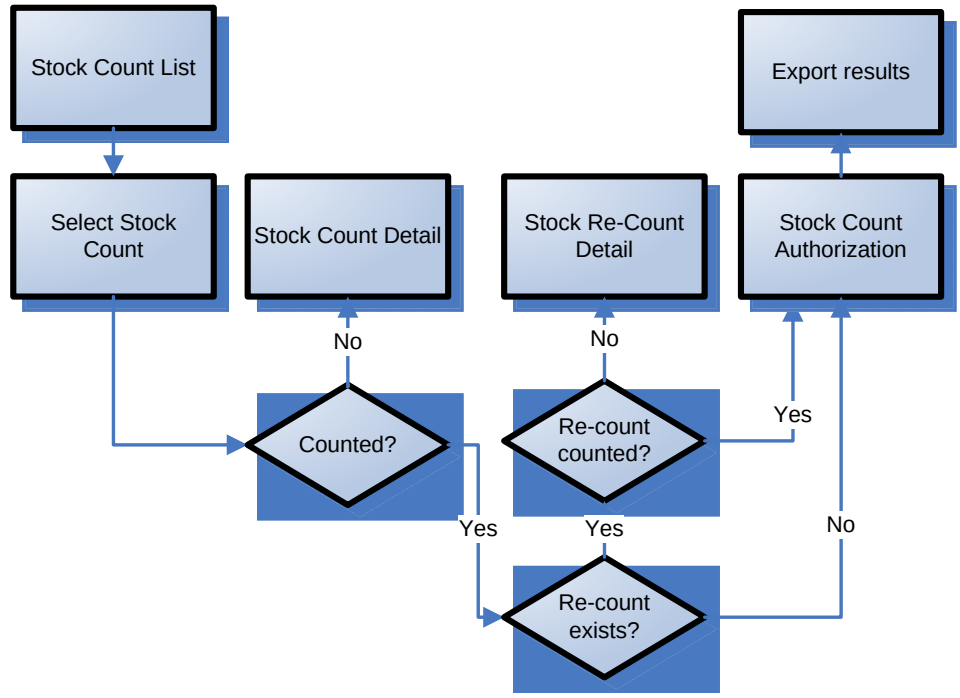
Third Party Unit & Amount



Business flow – Unit and amount all location



Business flow – Unit and amount



Shipping and Receiving Functional Overview

SIM has four distinctive shipping and receiving dialogues:

- 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
- Email alerts

The Creation of 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 merchandize system and reserve internally the inventory. This allows for a more accurate replenishment.

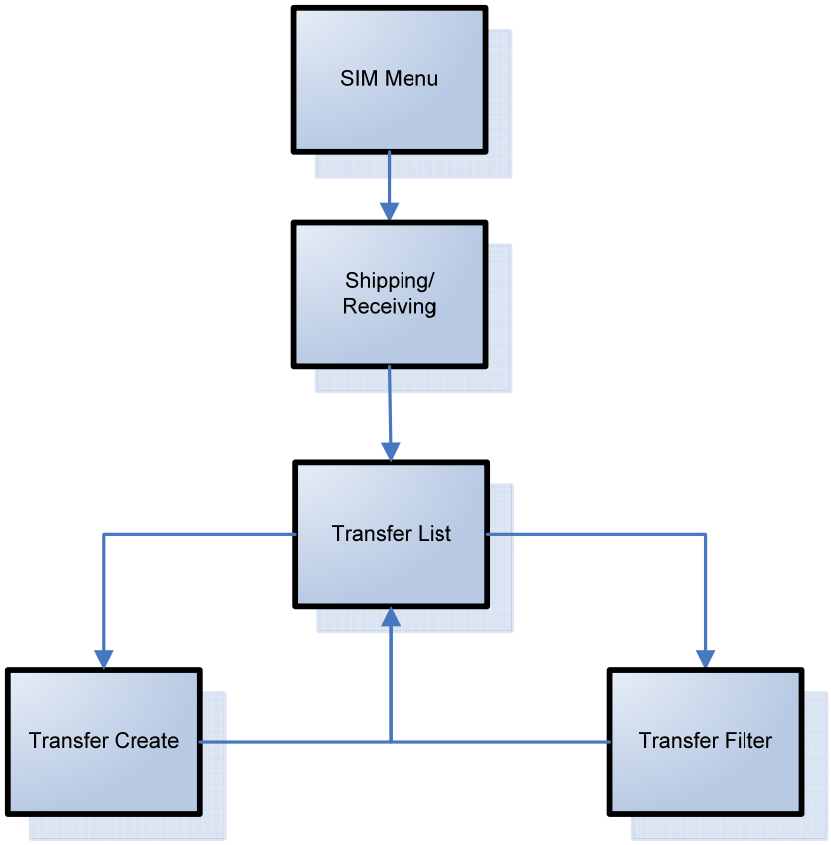
An Overview of Stock Movement after a Successful Dispatch

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.

Business Process Flow – Transfers PC



Transfer Requests

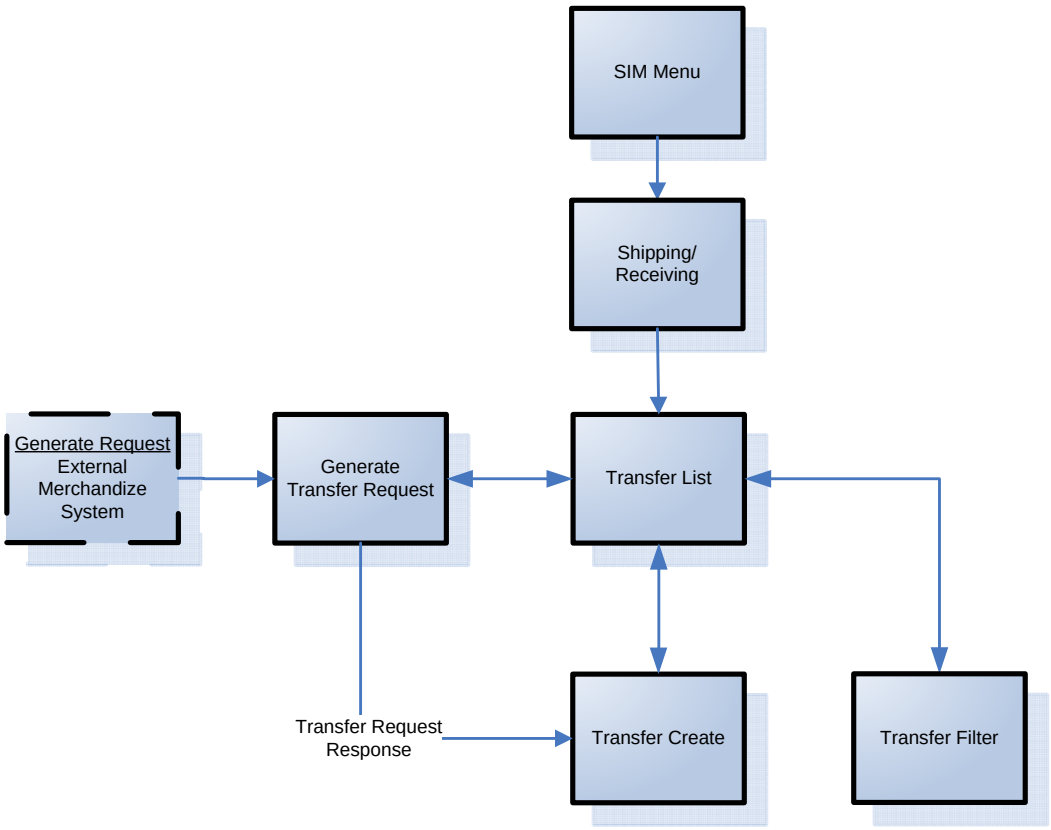
Transfer requests provide stores the ability to request products from other stores or allow corporate users to move inventory across stores using the central merchandising system. Transfer Requests are accessed from the Transfer dialogue. SIM allows for the lookup, creation, editing, and deletion of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

- Awaiting response
- Accepted
- Rejected
- Completed approved
- Completed rejected

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 email 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 dialogue where all of the items and quantities have been defaulted. From here on, the dialogue 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.

Business Process Flow – Transfer Requests PC



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.

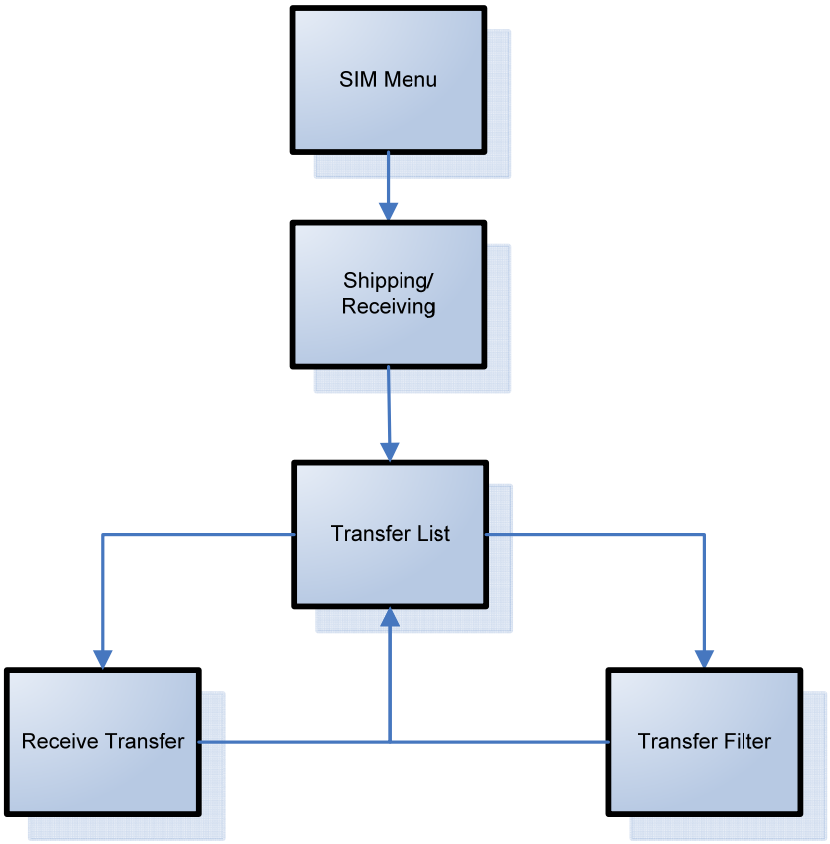
In the scenario where a transfer receipt being received is of substantial size and cannot be completed at one time, the user has the ability to save the delivery as **In Progress** status. This allows the user to save what has been received and return at a later time to continue receiving. Once the user has completed receipt of the entire 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 email 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.
- An auto receipt option exists for shipped transfers. This dialogue can be found under the administration section, SIM Stores, Auto-Receive stores.

Business Process Flow – Transfer Receiving PC



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.

Warehouse Delivery

Warehouse delivery functionality within SIM is utilized when goods are sent from a warehouse 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. The warehouses 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 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:

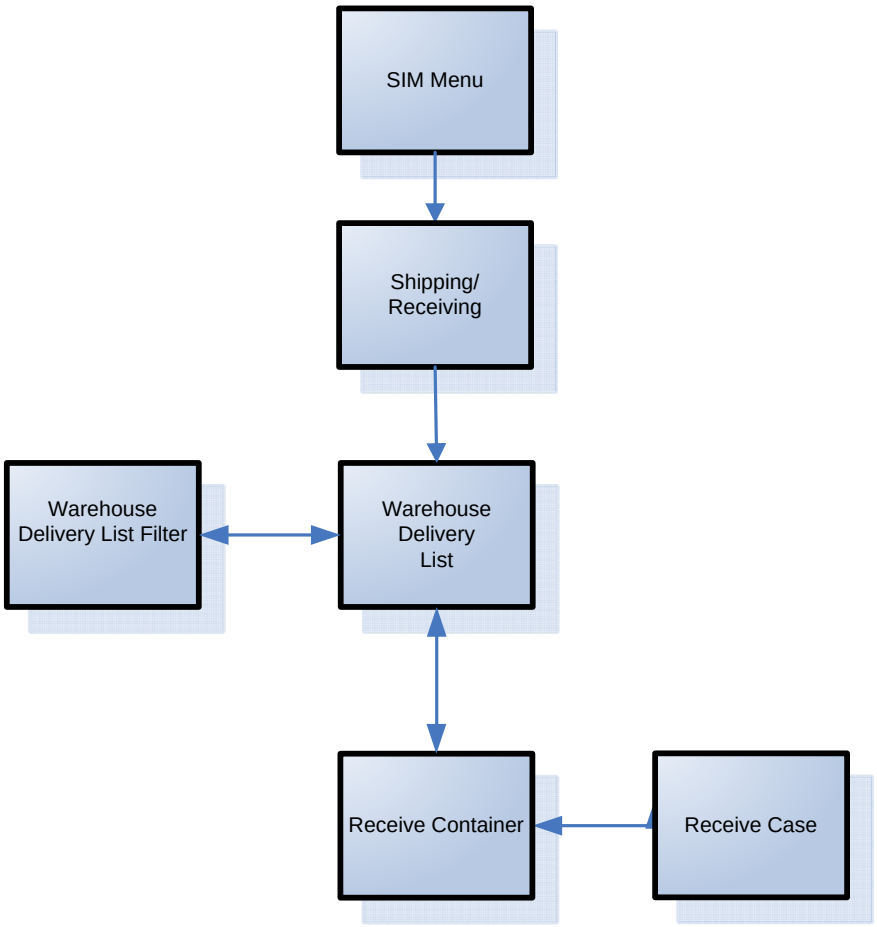
- Advanced Shipping Notice – This receiving level assumes a retailer's distribution center or warehouse system is very accurate and the store accepts the entire ASN without checking the content.
- Container - Store users can scan the barcode on the container (pallet/distribution unit/carton) within a 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 warehouse 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.

Business Process Flow – Warehouse Receiving 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 on an **Adjust Delivery** button 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.

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.

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.

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 merchandize 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.

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

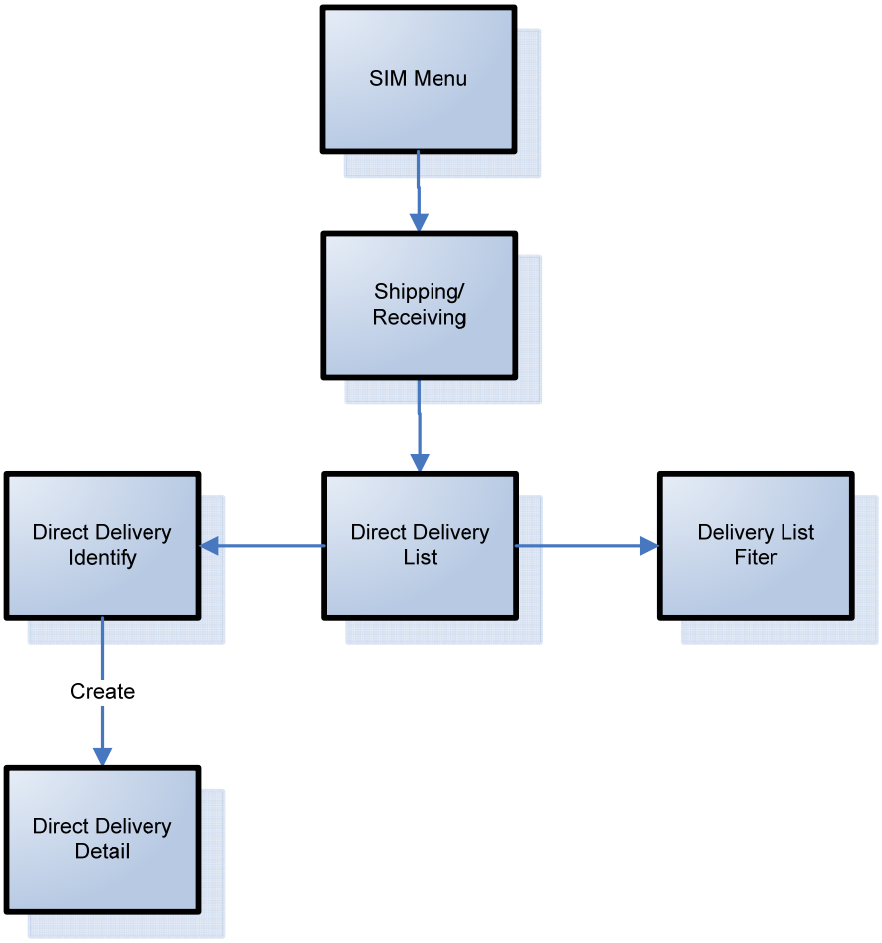
Direct Exchange (DEX) and Network Exchange (NEX) are uniform communications standards. DEX is the means through which a supplier, using a handheld device, can exchange electronic invoicing information with a store's direct store delivery (DSD) system. NEX differs in its delivery system, using the web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

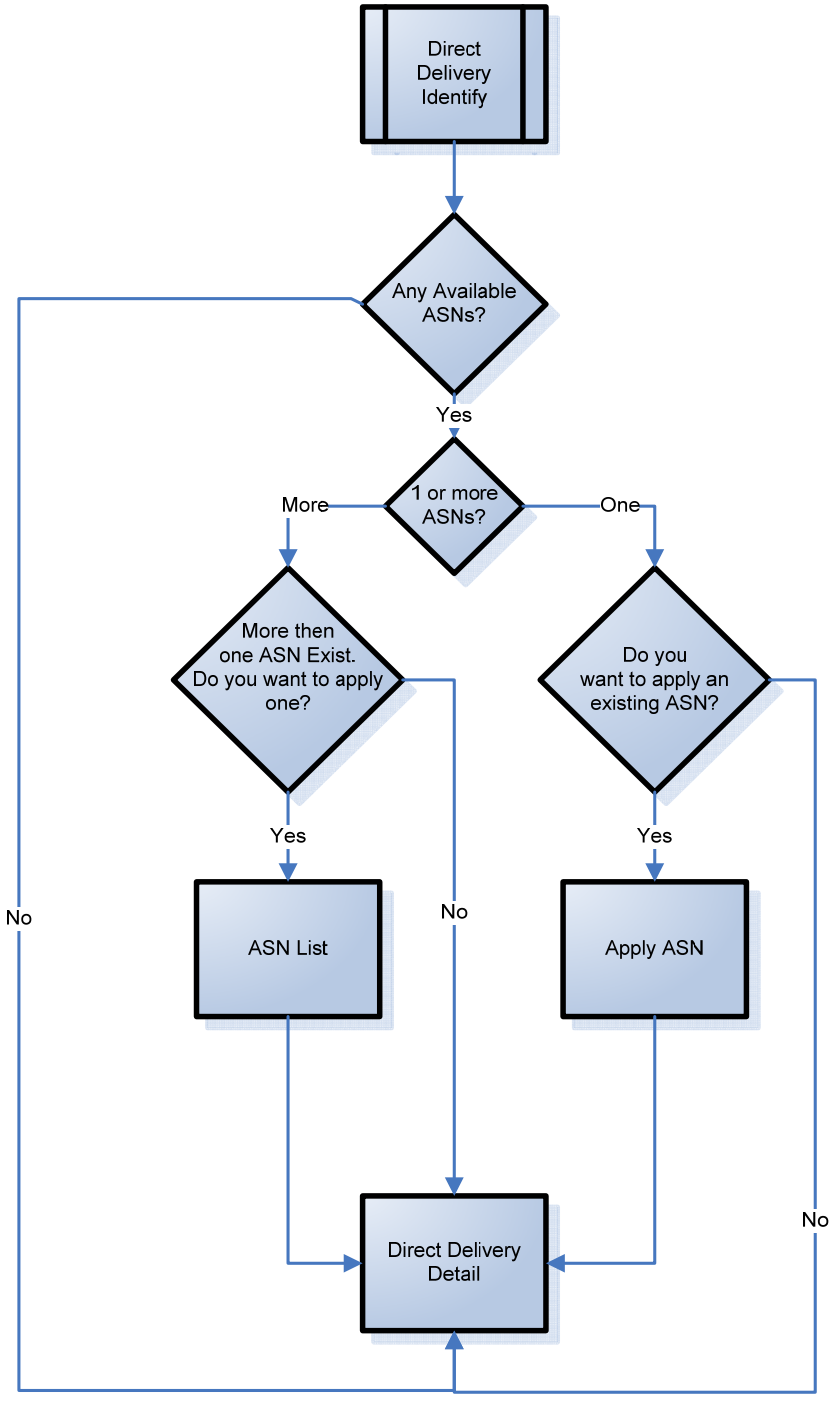
Existing POs vs. Quick Order Entry

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.

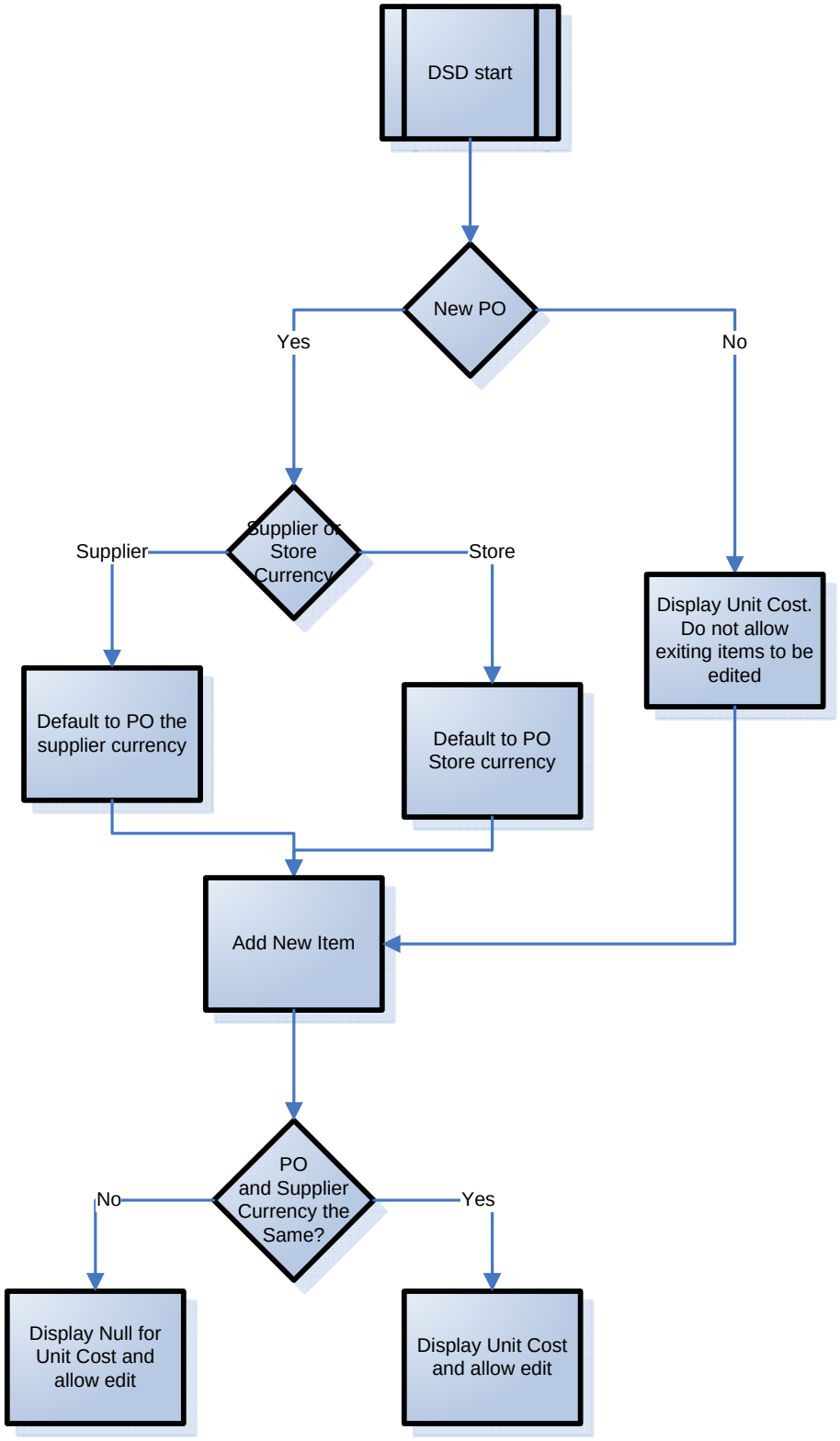
Business Process Flow – Direct Store Delivery PC (new created)



Business Process Flow – DSD Multiple Available ASNs



Business Process Flow - Updating and defaulting Cost



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 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 item(s) 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.

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. In that case, the user is asked whether he or she would like to use that for the return. 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 merchandize 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.

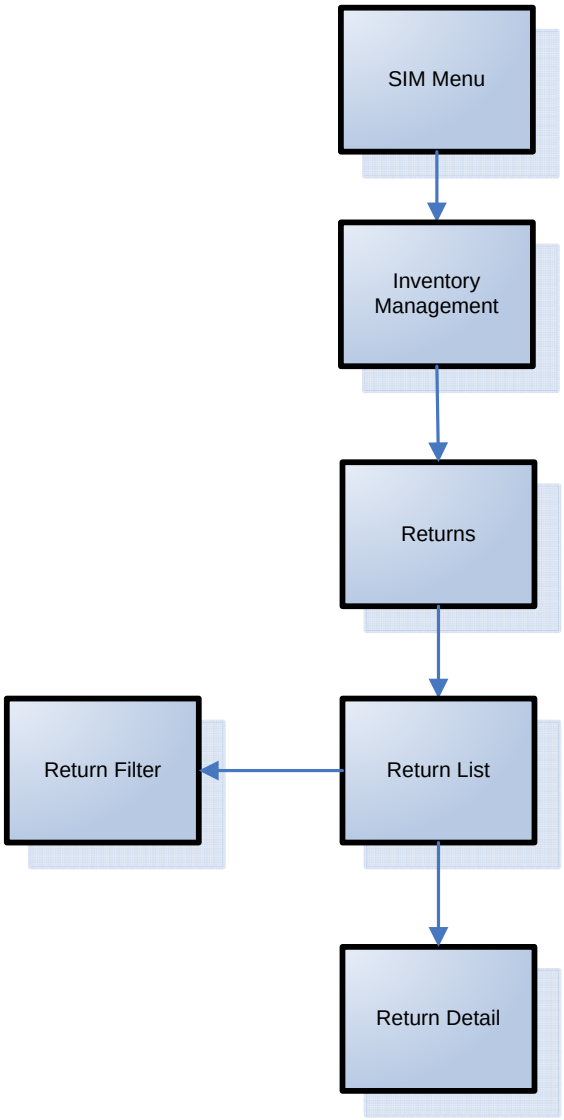
Return Requests

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.

Business Process Flow – Return 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:

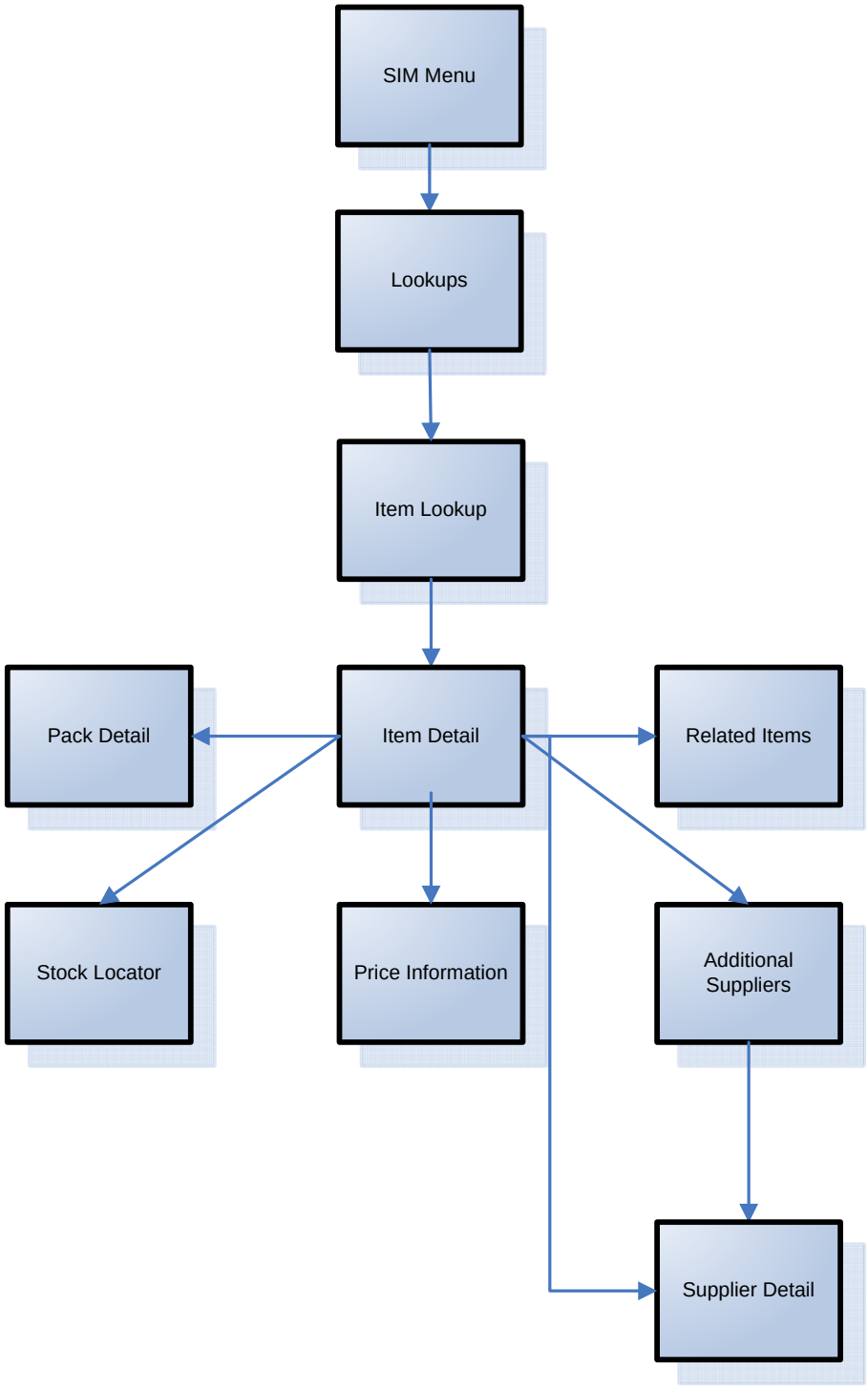
- Search Criteria
  - Item Number (UPC, SKU)
  - Item Description
  - (Primary) Supplier Name

  - (Primary) Supplier Number
  - Merchandize Hierarchy
  - Ranged indicator
- Information Displayed
  - Item Number (SKU)
  - Item Description
  - Supplier Name
  - Supplier Number
  - Primary UPC/EAN Number
  - Additional Suppliers
  - Item/Location ranging
  - Primary Sequence Location
  - Unit Of Measure
  - 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)
  - 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
  - Replenishment Method
  - Reject Store Orders – for store orders
  - Next Delivery Date- for auto replenishment items

In an effort to reduce the number of keystrokes, if a lookup on an item is done during the processing of a separate function (i.e. 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 has a very similar item. Example: The user presents a black large T-shirt, but the actually item wanted is a red medium sized T-shirt.

Business Process Flow – Item Lookup PC



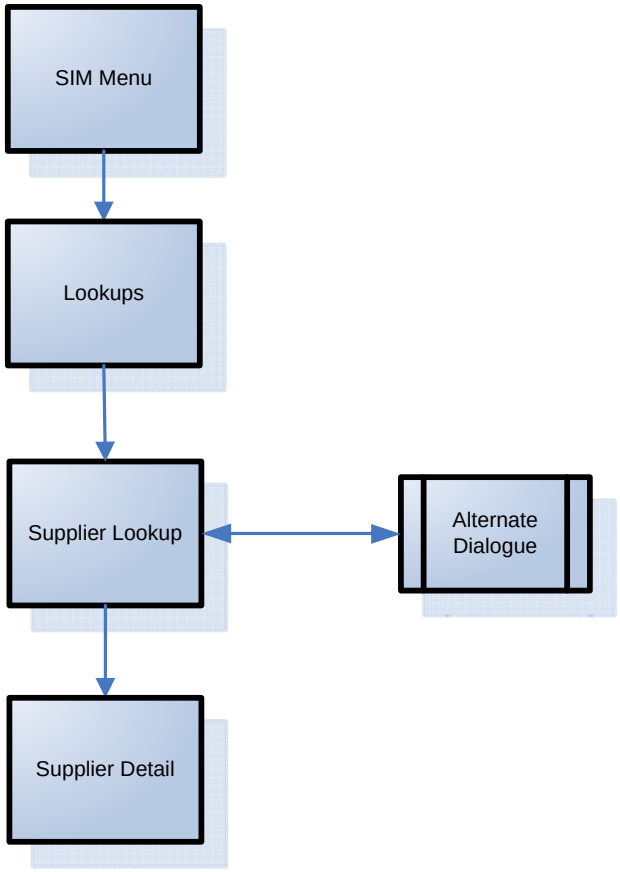
Supplier Lookup

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, Email Address
- Supplier Returns Address, Phone, Fax, Contact, Email 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 (i.e. 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.

Business Process Flow – Supplier Lookup PC



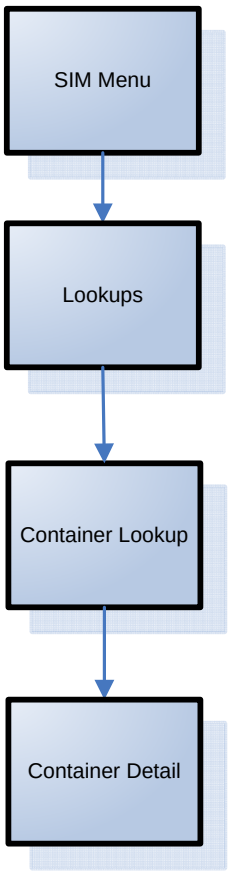
Container Lookup

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

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

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

Business Process Flow – Container Lookup PC



4

System and Store Administration

Overview

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 setup ad hoc stock count variance levels for each merchandize hierarchy at the class level.

- 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.

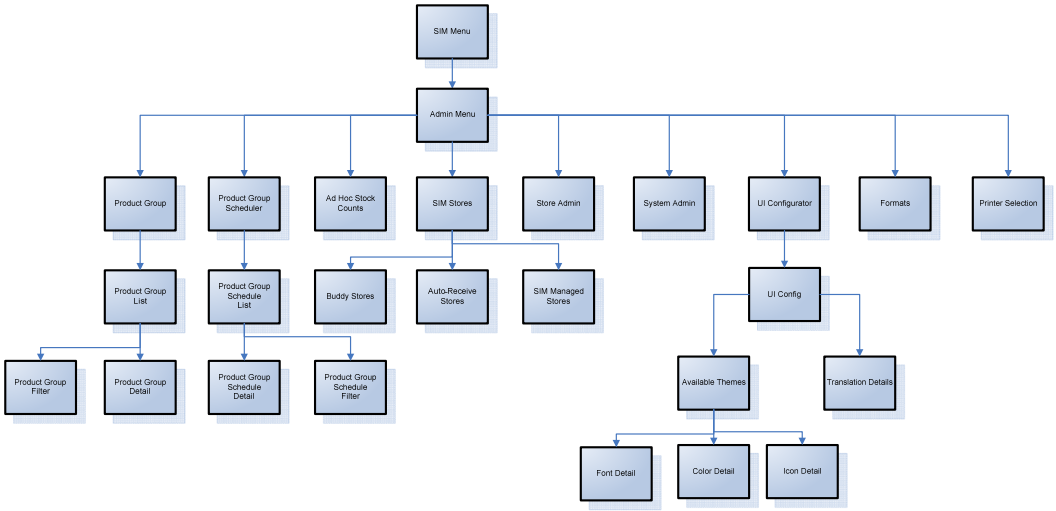
- UI Configuration

  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 dialogue as well.

- 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.



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
- Recounts
- Expiration
- Delivery dates

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. 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

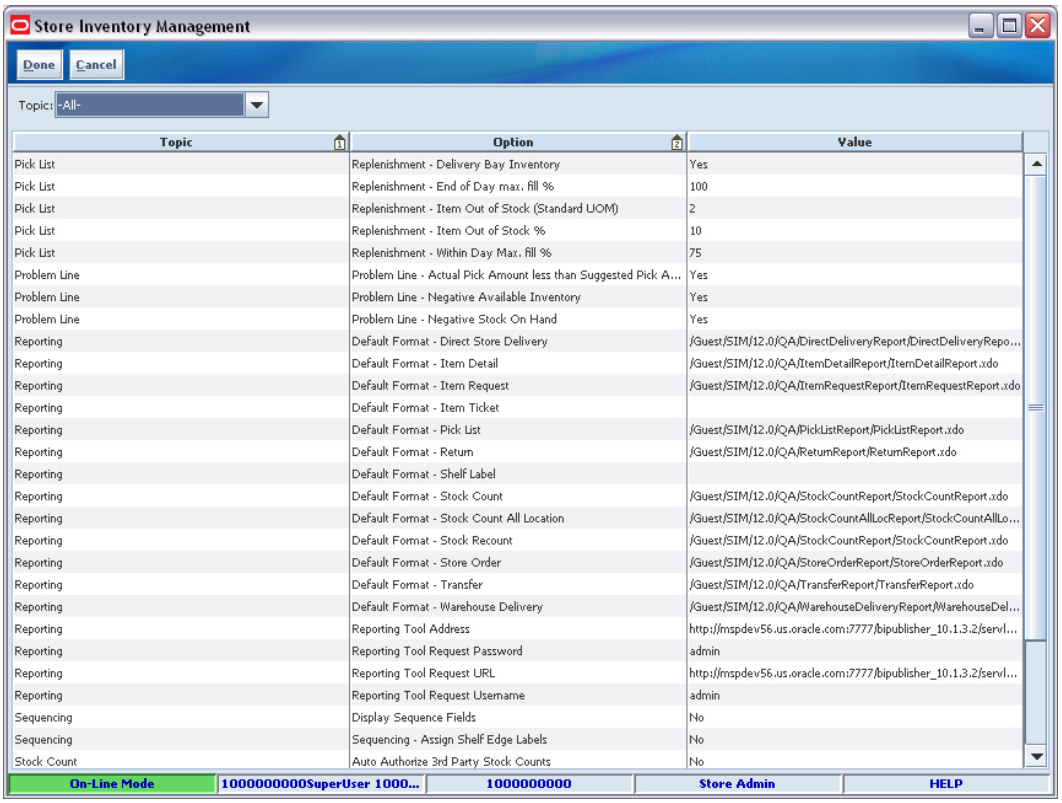
Store Administration

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

Navigate: Main Menu – Admin – Store Admin.

The Store Admin window opens.



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 drop-down list.
   - Enter an appropriate value in the field.
3. Click Done. You return to the Admin menu.

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

Store Administration Option	Valid Values	Default Value	Description
3rd Party Stock Counts – Unit and Amount	Yes, No	No	A value of Yes turns on third party unit and amount stock counts. This prevents the system from creating unit and amount stock counts that can be counted in the store.
Auto Authorize 3rd Party Stock Counts	Yes, No	No	When the unit and amount counted figures of the third party are uploaded, SIM provides the option to automatically authorize and approve the counted quantities.
Default Format – Direct Store Delivery	Text	DirectDeliveryReport	The value of this option indicates the default report that is printed when printing a direct store delivery report.
Default Format – Item Detail	Text	ItemDetailReport	
Default Format – Item Request	Text	ItemRequestReport	The value of this option indicates the default report that is printed when printing an item request.
Default Format – Item Ticket	Based on the tickets setup in SIM	(None)	The value of this option specifies the default format of the item ticket, when an item ticket is created within SIM (handheld or PC).
Default Format – Pick List	Text	PickListReport	The value of this option specifies the default report that is printed when printing a pick list report.
Default Format – Return	Text	ReturnReport11	The value of this option specifies the default report that is printed when printing a return report.
Default Format – Shelf Label	Based on the labels setup in SIM	(None)	The value of this option specifies the default format of the shelf edge label, when a shelf edge label is created within SIM (handheld or PC).
Default Format – Stock Count	Text	StockCountReport	The value of this option specifies the default report that is printed when printing a stock count report.
Default Format – Stock Count All Locations	Text	StockCountAllLocReport	

Store Administration Option	Valid Values	Default Value	Description
Default Format – Stock Count Not On File	Text	ThirdPartyStockCountNOFReport	
Default Format – Stock Recount	Text	StockCountRecountReport	The value of this option specifies the default report that is printed when printing a stock count recount report.
Default Format – Store Order	Text	StoreOrderReport	The value of this option specifies the default report that is printed when printing a direct store delivery report.
Default Format – Transfer	Text	TransferReport	The value of this option specifies the default report that is printed when printing a transfer report.
Default Format – Warehouse Delivery	Text	WarehouseDeliveryReport	The value of this option specifies the default report that is printed when printing a warehouse delivery report.
Display Sequence Fields	Yes, No	No	Indicates whether or not sequencing fields will be displayed in the Item Lookup screen.
Problem Line – Actual Pick Amount Less Than Suggested Pick Amount	Yes, No	Yes	If the value of this option is Yes, when the problem line batch program runs, any item for which the actual pick amount was less than the suggested pick amount is added to the problem line stock count.
Problem Line – Negative Available Inventory	Yes, No	Yes	If the value of this option is Yes, when the problem line batch program runs, any item that has negative available inventory is added to the problem line stock count.
Problem Line – Negative Stock on Hand	Yes, No	Yes	If the value of this option is Yes, when the problem line batch program runs, any item that has negative stock on hand is added to the problem line stock count.
Replenishment – Delivery Bay Inventory	Yes, No	Yes	This option allows you to turn on or off the delivery bay functionality.

Store Administration Option	Valid Values	Default Value	Description
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.
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.
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.
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 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 Tool Request Password	Text	(None)	
Reporting Tool Request URL	Text	(None)	This option can specify the URL of the specific report request. The report is printed without any user interface with the Crystal Reports server.
Reporting Tool Request Username	Text	admin	
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.

Store Administration Option	Valid Values	Default Value	Description
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.
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.
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.
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.
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.

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.

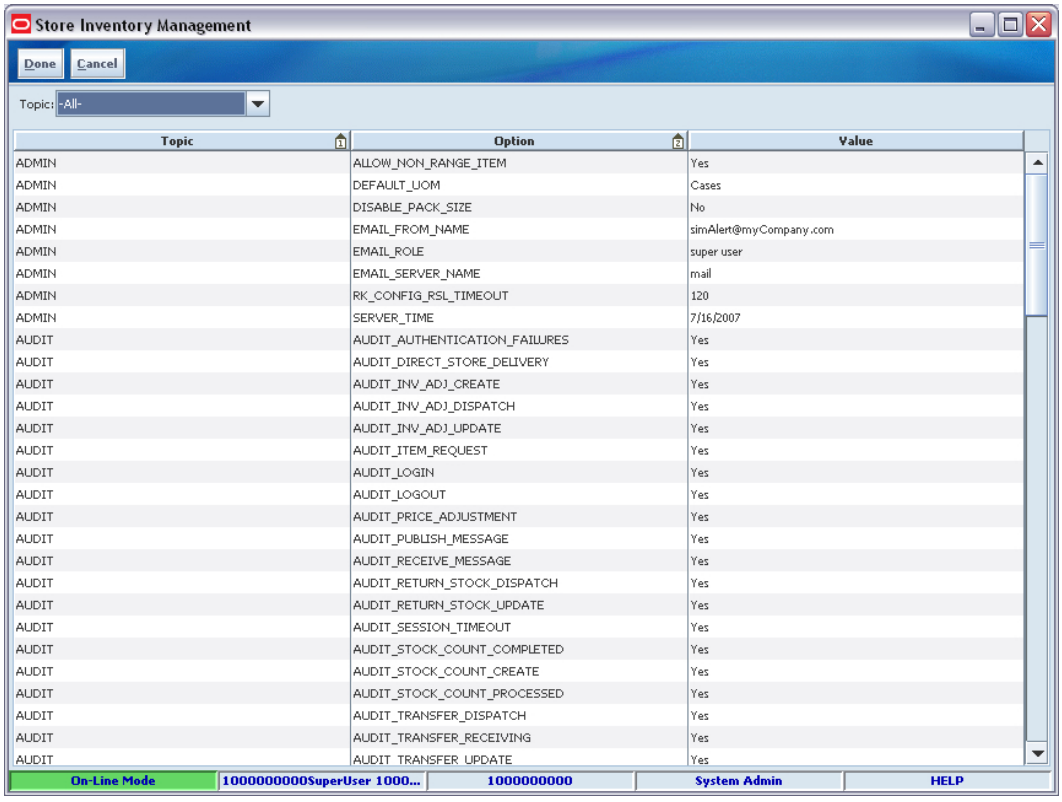
The system options are in these general categories:

- 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

Navigate: Main Menu – Admin – System Admin.

The System Admin window opens.



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 drop-down list.
   - Enter an appropriate value in the field.
3. Click Done. You return to the Admin menu.

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

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	30	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.

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 ID, Supplier ID, 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	
Direct Delivery Send Null Unit Cost	Yes, No	No	
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.
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 Pack Receiving in Direct Deliveries	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.

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 Email 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.

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	100	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 Alerts	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.

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. NL: 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. SL: 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. RL: 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. **Note:** 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 financials 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 Email Alert	Yes, No	Yes	If this option is set to Yes, an e-mail alert is sent when a transfer request has been rejected.

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.

System Administration Option	Valid Values	Default Value	Description
Default UOM	Cases, Standard UOM	Cases	This option allows the store to select the default unit of measure that the store will normally use. In most of the functional areas of the application, the selected option defaults. If the case quantity is not a whole number when it is initially displayed on the screen, the standard UOM is displayed as the default, regardless of this setting.
Disable Pack Size	Yes, No	No	This option allows the user to edit the pack size in the application.
Email From Name	Text	simAlert@ myCompany.com	When the system sends e-mail alerts, the specified e-mail address is displayed in the 'from' name.
Email Role	Text	Super User	When the system sends e-mail alerts, the alerts are sent to the specified user role.
Email Server Name	Text	mail	This option specifies the server name used for e-mail alerts.
RSL Timeout (Seconds)	Numeric	120	This option specifies the number of seconds before a timeout message is returned.

Audit Options

System Administration Option	Valid Values	Default Value	Description
Audit Authentication Failures	Yes, No	Yes	SIM checks each of the other audit options (parameters) to determine whether those transactions need to be logged. This audit option uses process ID 2 (Login_unsuccessful) with the following key values: 1 - Invalid password 2 - Store authorization not found 3 - Rule violation 4 - Invalid login ID
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).

System Administration Option	Valid Values	Default Value	Description
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 Request	Yes, No	Yes	Completed item requests are tracked with process ID 201 (Store_Order_request).
Audit Login	Yes, No	Yes	Users' successful logins to SIM are tracked with Process ID 1 (Login_successful).
Audit Logout	Yes, No	Yes	Users' logouts from SIM are tracked with process ID 3 (Log_off).
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).
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).

System Administration Option	Valid Values	Default Value	Description
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.

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 Adjustment	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	
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.

System Administration Option	Valid Values	Default Value	Description
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.
Enable GMT for Third Party Stock Count	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.

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 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.

System Administration Option	Valid Values	Default Value	Description
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 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 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.
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	10	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.

System Administration Option	Valid Values	Default Value	Description
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 Shipments	Numeric	30	All records in 'Complete' and 'Cancelled' 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 'Cancelled' 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

System Administration Option	Valid Values	Default Value	Description
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.
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.

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.

User Interface Options

System Administration Option	Valid Values	Default Value	Description
Display handheld Length: Diff 1	Numeric	6	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.

System Administration Option	Valid Values	Default Value	Description
Display handheld Length: Diff 2	Numeric	6	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 handheld Length: Diff 3	Numeric	6	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 handheld Length: Diff 4	Numeric	6	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	Long 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 Stock Locator	Yes, No	Yes	This option allows the user to access the Stock Locator screen from Item Detail on the PC. If this option is set to Yes, the Stock Locator button is displayed; otherwise it is not displayed.
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.

System Administration Option	Valid Values	Default Value	Description
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.
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.
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.

Stock Count Options

System Administration Option	Valid Values	Default Value	Description
Display Update Auth Qty Button on Stock Count Authorize Screen	Yes, No	Yes	To prevent semi-automatic validation without real review, the Update Auth Qty button in the Stock Count Authorization window can be turned off.
Print One Stock Count Page per Location	Yes, No	No	This option allows the user to print stock counts by location, with one page equaling one location.
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.

System Administration Option	Valid Values	Default Value	Description
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 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.
Unguided Stock Counts – 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.
Unit and Amount Stock Count Use Guided Handheld	Yes, No	No	This option determines whether or not the handheld device guides the user through unit and amount stock counts based on sequencing (or item number if sequencing is not being used).
Unit Stock Count Use Guided Handheld	Yes, No	No	This option determines whether the handheld device guides the user through unit stock counts by sequence (or item number if not sequenced), or does not prompt the user at all.

Pricing Options

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

Reporting Options

System Administration Option	Valid Values	Default Value	Description
Enable Report Printing Failure Override	Yes, No	No	This indicator will prompt the user to continue or not if printing fails. If the option is turned on and the printer option fails, the user will be prompted with a message asking them to continue or abort. If the option is turned off, the user will not get the prompt but rather the printer error message we see today. In case of stock count reports and the stock recount reports the printing results in change of status of the stock count. In case of other reports there is no impact of the printing of the report. Hence the impact of this parameter will only be in case of the reports related to stock counts and stock recounts. In case of all other reports the system will behave as it would in case the option is turned off.
Report Failure Error Handling	Yes, No	Yes	

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.

5

Reporting

Overview

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 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).

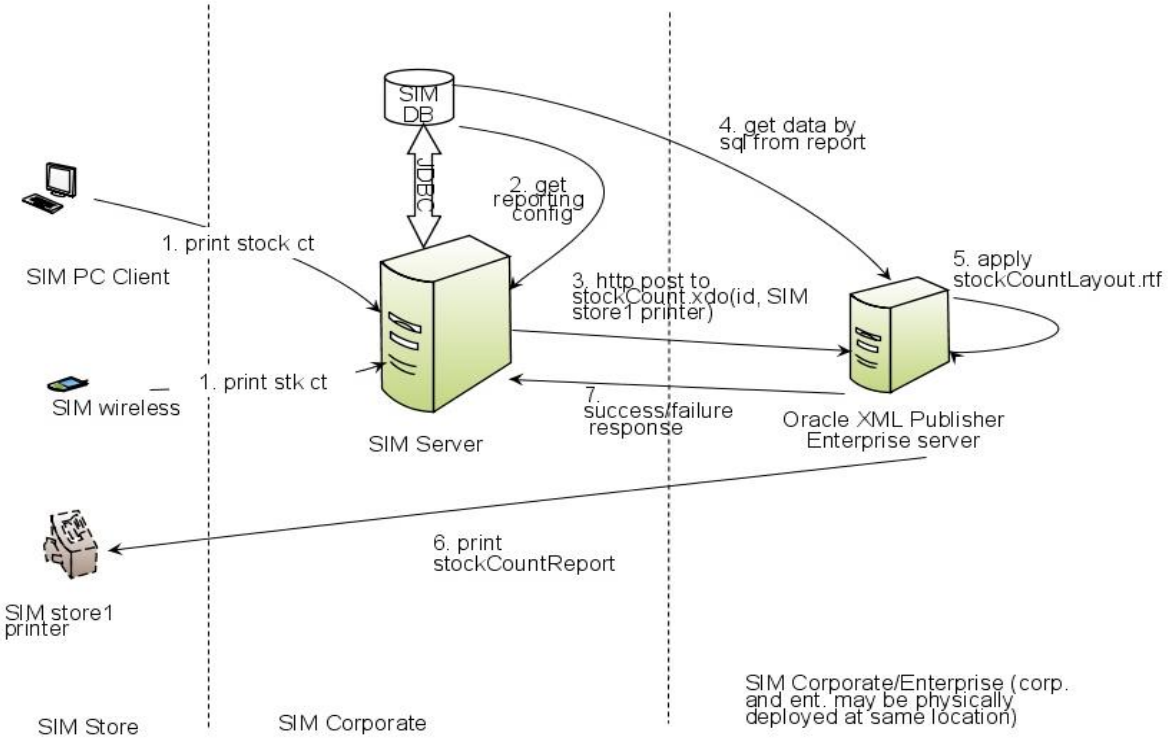
Assumptions

- 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

The sequence of operations for a print request is as follows:

1. The user presses the print button on the SIM interface (PC or wireless).
2. SIM gets the printer selection from the user, if it is the first print request in the session. For subsequent print requests, the printer defaults to the last selected printer.
3. SIM builds a report request comprising standard parameters such as logical name of the report, selected printer name, locale and store id in addition to any other report specific parameters (like Stock Count Id, etc).
4. Using these request parameters, SIM constructs a Report request and routes the request via http to the pre-configured Reporting Tool Request URL (which points to the BI Publisher installation URL).
5. BI Publisher identifies the report, queries the database to get the data needed for the report, formats the data, and sends the report to a destination (in this case a logical name of a printer that is pre-configured on BI publisher).
6. BI Publisher responds with a success or failure message. The response is only an indication of report success or failure, meaning the data for reporting is available and report formatting was successful. Any print failures are reported on BI Publisher's scheduler log.



SIM Operational Reports

Report Name	Report Parameters	Report view/s
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
StockCountReport	STOCK_COUNT_ID, Store_Timezone	STOCK_COUNT_REPORT_V
StockCountRecountReport	STOCK_COUNT_ID, Store_Timezone	STOCK_COUNT_REPORT_V
ItemDetailReport	ITEMID, STOREID, Store_Timezone	ITEM_DETAIL_REPORT_V, ITEMDET_SEQUENCE_V, ITEMDET_ALLOCATION_V
WarehouseDeliveryReport	Document_ID, Store_Timezone	WAREHOUSE_DELIVERY_REPORT_V, WAREHOUSE_DELIVERY_SHIPMENT_V
TransferReport	Transfer_ID, Store_Timezone	TRANSFER_REPORT_V
StockCountAllLocReport	STORE_ID, STOCK_COUNT_ID	STOCK_COUNT_REPORT_V
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

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 a .xdo file and a .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.

Setting up SIM

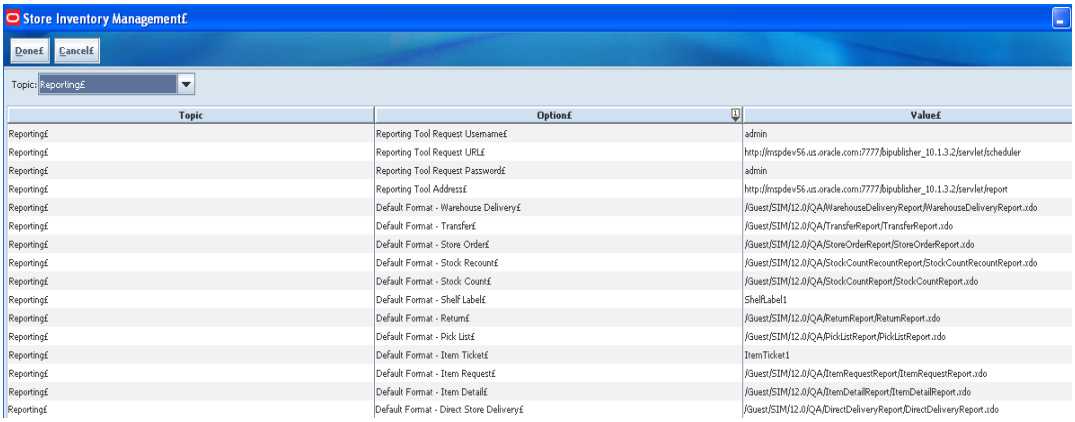
Select the Reporting topic on the SIM Store Admin Config screen. The following options need to be set up:

Option	Value
Reporting Tool Request User name	*<BIP_REPORTS_USER>*
Reporting Tool Request User password	*<BIP_REPORTS_USER_PASSWORD>*
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 the 'Reports' button is pressed on the SIM PC client.
Default Format – Warehouse Delivery	/*<BIP_SIM_REPORTS_FOLDER>*/WarehouseDeliveryReport/WarehouseDeliveryReport.xdo
Default Format – Transfer	/*<BIP_SIM_REPORTS_FOLDER>*/TransferReport/TransferReport.xdo
Default Format – Store Order	/*<BIP_SIM_REPORTS_FOLDER>*/StoreOrderReport/StoreOrderReport.xdo
Default Format – Stock Recount	/*<BIP_SIM_REPORTS_FOLDER>*/StockCountRecountReport/StockCountRecountReport.xdo
Default Format – Store Count	/*<BIP_SIM_REPORTS_FOLDER>*/StockCountReport/StockCountReport.xdo
Default Format – Return	/*<BIP_SIM_REPORTS_FOLDER>*/ReturnReport/ReturnReport.xdo
Default Format – Item Request	/*<BIP_SIM_REPORTS_FOLDER>*/ItemRequestReport/ItemRequestReport.xdo
Default Format – Item Detail	/*<BIP_SIM_REPORTS_FOLDER>*/ItemDetailReport/ItemDetailReport.xdo
Default Format – Direct Store Delivery	/*<BIP_SIM_REPORTS_FOLDER>*/DirectDeliveryReport/DirectDeliveryReport.xdo
Default Format – Item Ticket	Can only select from a list of options. To populate the actual formats go to the Formats tab and add a new format with /*<BIP_SIM_REPORTS_FOLDER>*/ItemTicketReport1/ItemTicketReport1.xdo in the FORMAT NAME. You can add multiple formats for Item Tickets.
Default Format – Shelf Label	Can only select from a list of options. To populate the actual formats go to the Formats tab and add a new format with: /*<BIP_SIM_REPORTS_FOLDER>*/ShelfLabelReport1/ShelfLabelReport1.xdo in the FORMAT NAME. You can add multiple formats for a Shelf Label.

Notes:

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

<BIP_SIM_REPORTS_FOLDER> is the folder where SIM reports have been uploaded on the BI Publisher server. For example, if they have been uploaded in the Guest folder, it is /Guest.



Topic	Option£	Value£
Reporting£	Reporting Tool Request Username£	admin
Reporting£	Reporting Tool Request URL£	http://mspdev56.us.oracle.com:7777/bipublisher_10.1.3.2/servlet/scheduler
Reporting£	Reporting Tool Request Password£	admin
Reporting£	Reporting Tool Address£	http://mspdev56.us.oracle.com:7777/bipublisher_10.1.3.2/servlet/report
Reporting£	Default Format - Warehouse Delivery£	/Guest/SIM/12.0/QA/WarehouseDeliveryReport/WarehouseDeliveryReport.xdo
Reporting£	Default Format - Transfer£	/Guest/SIM/12.0/QA/TransferReport/TransferReport.xdo
Reporting£	Default Format - Store Order£	/Guest/SIM/12.0/QA/StoreOrderReport/StoreOrderReport.xdo
Reporting£	Default Format - Stock Recount£	/Guest/SIM/12.0/QA/StockCountRecountReport/StockCountRecountReport.xdo
Reporting£	Default Format - Stock Count£	/Guest/SIM/12.0/QA/StockCountReport/StockCountReport.xdo
Reporting£	Default Format - Shelf Label£	ShelfLabel1
Reporting£	Default Format - Return£	/Guest/SIM/12.0/QA/ReturnReport/ReturnReport.xdo
Reporting£	Default Format - Pick List£	/Guest/SIM/12.0/QA/PickListReport/PickListReport.xdo
Reporting£	Default Format - Item Ticket£	ItemTicket1
Reporting£	Default Format - Item Request£	/Guest/SIM/12.0/QA/ItemRequestReport/ItemRequestReport.xdo
Reporting£	Default Format - Item Detail£	/Guest/SIM/12.0/QA/ItemDetailReport/ItemDetailReport.xdo
Reporting£	Default Format - Direct Store Delivery£	/Guest/SIM/12.0/QA/DirectDeliveryReport/DirectDeliveryReport.xdo

Report Engine Functional Specification

Functional Overview

The reporting functionality has been enhanced to incorporate error handling when reports are printed. Error handling allows the user to continue in the event that the printing effort fails. Without error handling, printing failures can cause problems in the store. Sometimes the report information is crucial, but sometimes it is not or the information can be printed after the event by a custom report.

To prevent these bottlenecks, SIM provides a prompt that allows you to continue if the printer fails.

Functionality Checklist

Description of Modification	New Screen / Process	Existing Screen / Process
Add system option	X	
Add Print button on item detail		X
Add Print option on item lookup handheld		X
Print Report	X	
Upgrade to XML Publisher	X	

Functional Requirements

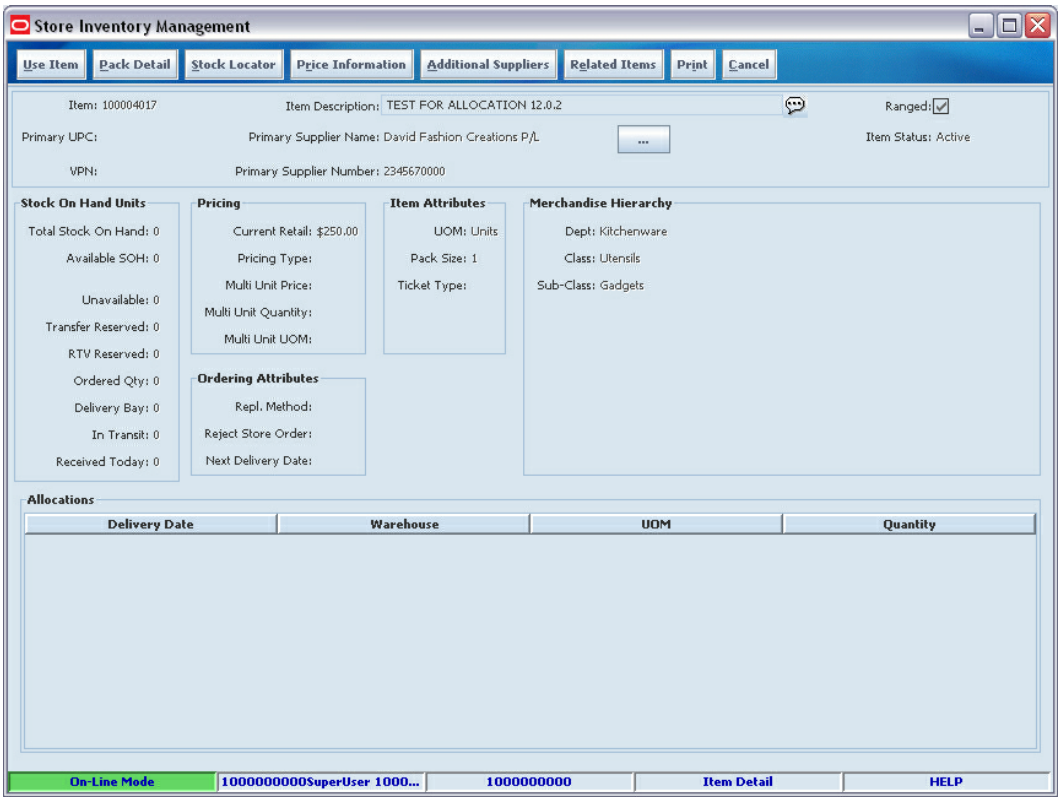
System Parameters

Report Failure Error Handling

- Name: Enable report printing failure override
- Value: Yes/no
- A system option prompts the user to continue or not if printing fails.
- If the option is turned on and the printer option fails, the user is prompted with a message asking them to continue or abort.
- If the option is turned off, the user does not get the prompt but rather a printer error message.
- In case of stock count reports and the stock recount reports, the printing results in a change of status of the stock count. In case of other reports there is no impact of the printing of the report. Hence the impact of this parameter will only be in the case where the reports related to stock counts and stock recounts.
- In case of all other reports, the system will behave as it would in case the option is turned off.

Item Lookup Detail Screen - PC

Screen Flow



Screen Shot

Printing Reports

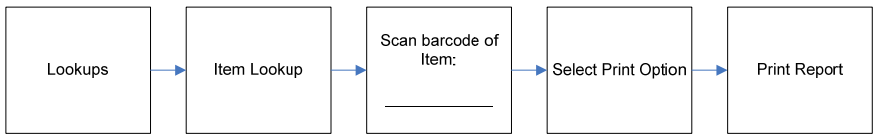
Use the Print button on the Item Detail screen to print reports. Pressing the Print button when only one item has been selected prints the new item report for that item.

SIM Report List

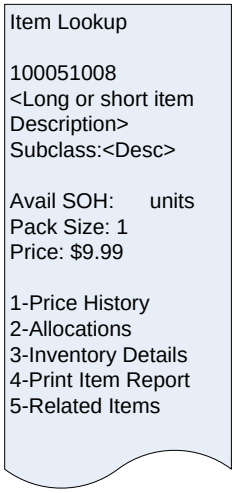
The following reports have been converted to the new Reports engine:

- Direct Delivery
- Item Request
- Pick List
- Warehouse Delivery
- Returns
- Stock Count / Stock Recount
- Store Order
- Transfers
- Item Lookup

Item Lookup Detail Screen - Handheld



Screen Flow Overview



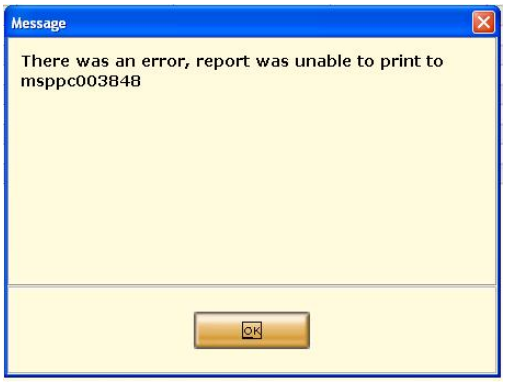
Item Lookup Screen Shot

Error Handling – Report Printing

General Requirements

The error handling for report printing is based on a system option "Enable report printing failure override".

- If the system option is set to "Yes", the report printing displays the following error message in case the print job fails: "There was an error, the report was unable to print to <*Printer name*>. Do you want to continue?"
  - Selecting the "Yes" button allows the regular SIM processing to continue.
    Example: When printing a stock count on the PC, the snapshot is taken. Pressing the yes button will continue taking the snapshot.
  - Selecting the "No" button stops processing and bring the user back to the screen where the print button was pressed.
    It should work in the same way as if the "Enable report printing failure override" option is set to "No" and the user pressed the OK button.
- If the system option is set to "No", the report printing displays the existing error message in case the print job fails: "There was an error; the report was unable to print to <*Printer name*>."
  - Pressing the OK button brings the user back to the screen where they pressed the print button.



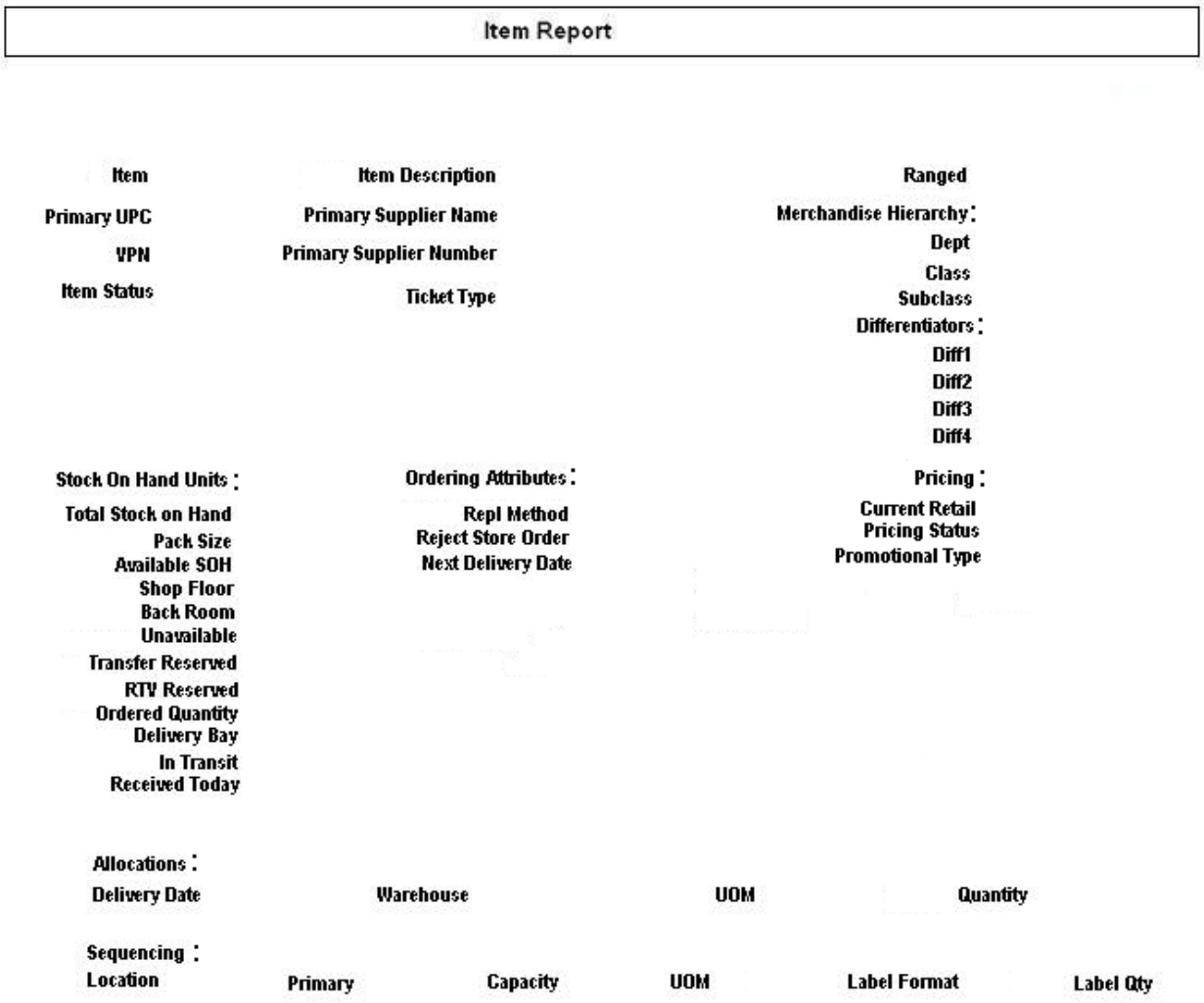
Example of Error Message when "Enable report printing failure override" Is Set to "No"

Detailed Report information

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)



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.

The report is based on the view DIRECT_DELIVERY_REPORT_V

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:

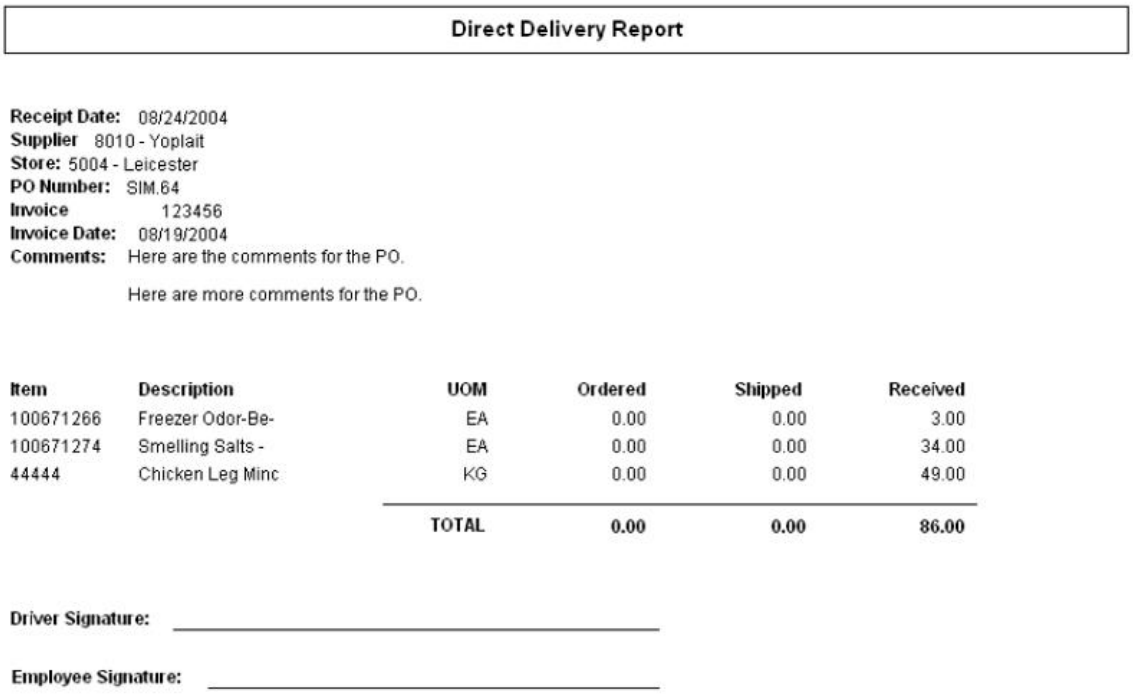
- Item ID – Item number for each line item received
- Item Description – Description of item
- Unit of Measure – Unit of measure for quantity (Cases or Eaches)
- Quantity Ordered – Quantity ordered according to the PO
- Quantity Shipped – Quantity shipped according to the shipment record
- Quantity Received – 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.

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 is based on the view ITEM_REQUEST_REPORT_V

The report consists of two sections with the following information:

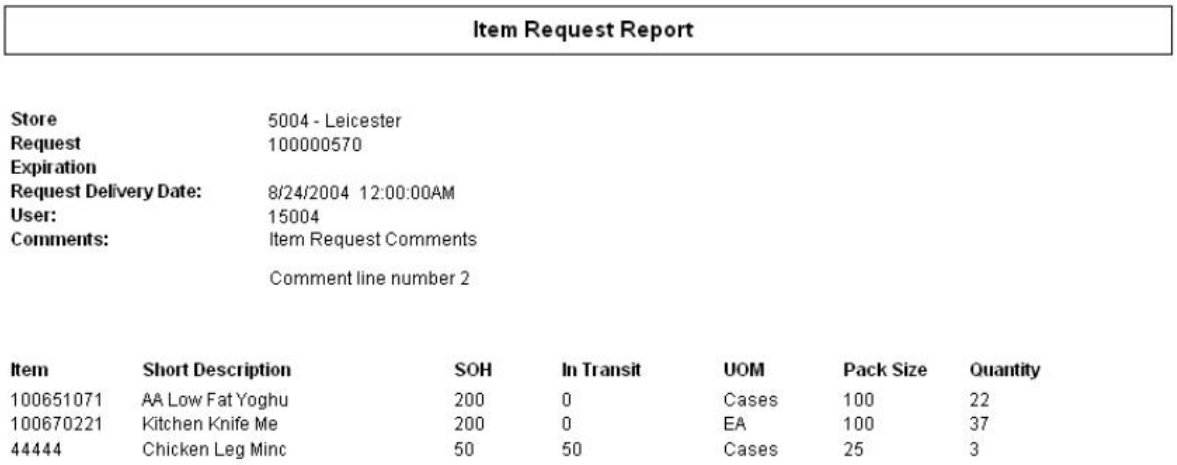
Header:

- 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

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 is based on the view PICK_LIST_REPORT_V

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 backroom 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

Pick List Report

Pick List Report

ID: 43
Product Group: Pick List Home Shop
Create Date/Time: 8/26/2004 2:27:49PM
User: 15004
Status: COMPLETE
Quantity: 500

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 is based on the view WAREHOUSE_DELIVERY_REPORT_V

The report consists of two sections.

Header:

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

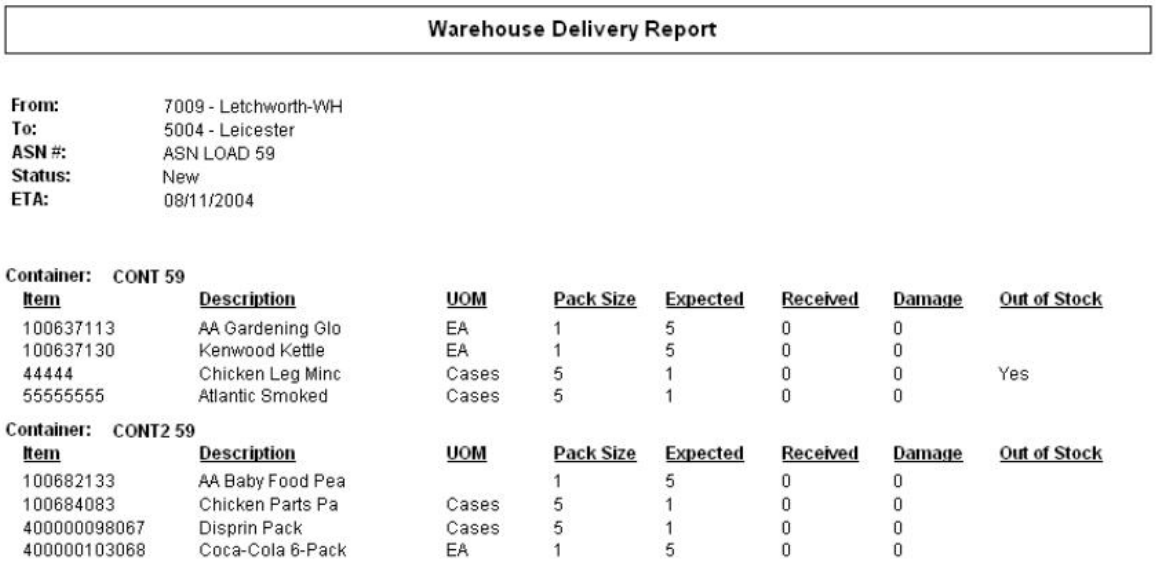
- From – Originating Warehouse location details
- To – Destination store location details
- ASN # - Identifier for the shipment being received
- Status – Status of the warehouse delivery
- ETA – Expected arrival date of the warehouse delivery

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

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 is based on the view RETURN_REPORT_V

The report consists of two sections with the following information:

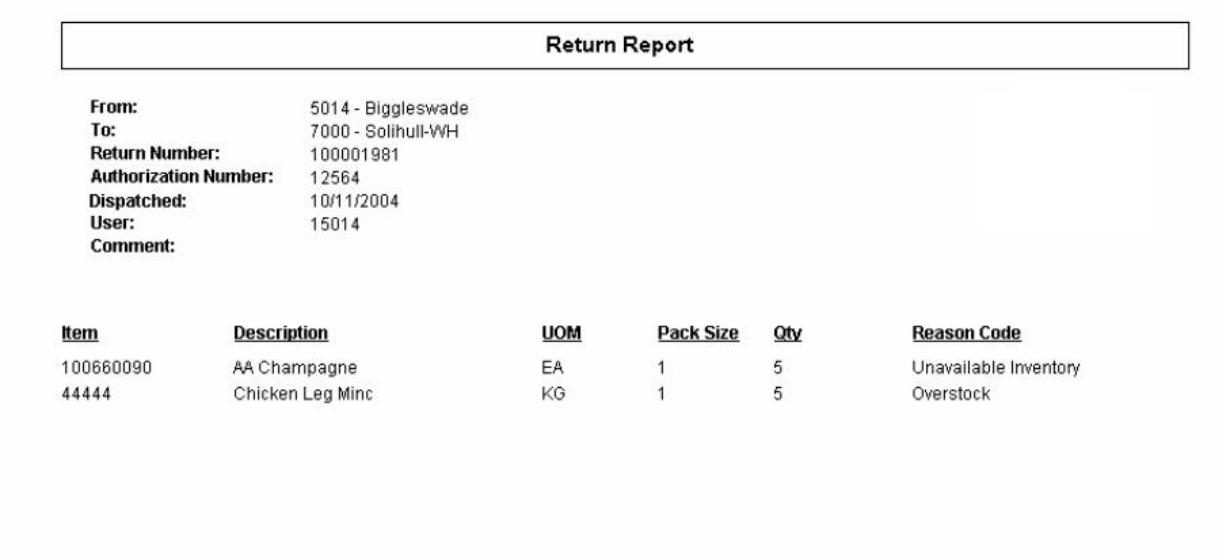
Header:

- 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

Return Report



Return Report

From: 5014 - Biggleswade
To: 7000 - Solihull-WH
Return Number: 100001981
Authorization Number: 12564
Dispatched: 10/11/2004
User: 15014
Comment:

Item	Description	UOM	Pack Size	Qty	Reason Code
100660090	AA Champagne	EA	1	5	Unavailable Inventory
44444	Chicken Leg Minc	KG	1	5	Overstock

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 is based on the view STOCK_COUNT_REPORT_V

The report consists of two sections with the following information:

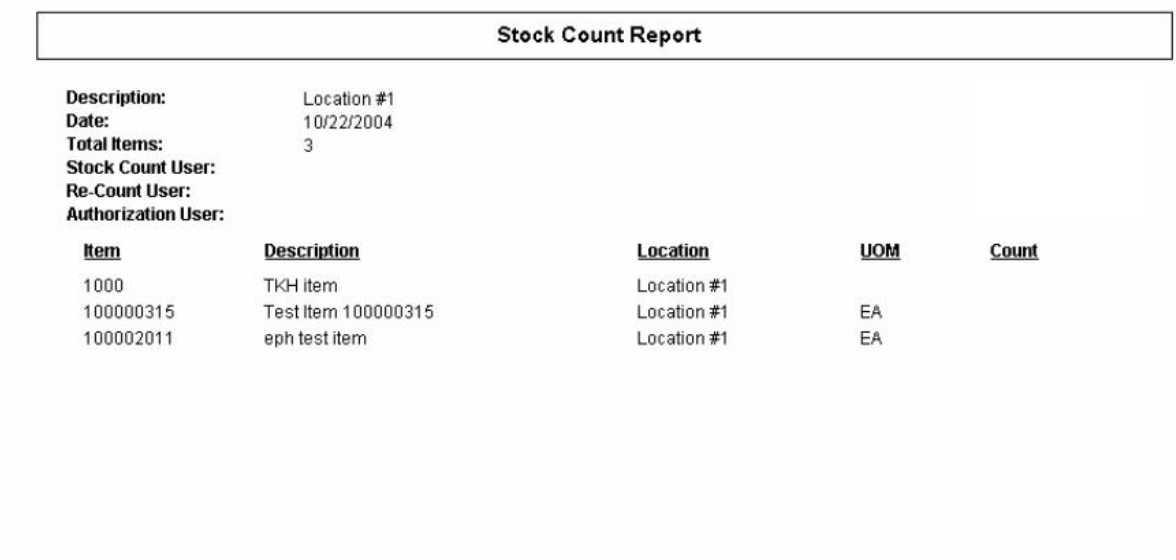
Header:

- Description – Stock count description
- Date – Scheduled date for the stock count
- Total Items – Total number of items in the stock count
- Stock count user – User who created the stock count
- Recount user – User who requested the recount
- Authorization user – User who authorized the stock count

Detail:

- Item – Item number for each line item in the stock count
- Description - Description of item
- Location – Sequenced location in the store being counted
- UOM – Unit of measure for the item
- Count – Physical count results entered for the stock count

Stock Count Report



Stock Count Report

Description: Location #1
Date: 10/22/2004
Total Items: 3
Stock Count User:
Re-Count User:
Authorization User:

Item	Description	Location	UOM	Count
1000	TKH item	Location #1		
100000315	Test Item 100000315	Location #1	EA	
100002011	eph test item	Location #1	EA	

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 is very similar to the stock count report and is based on the same view STOCK_COUNT_REPORT_V.

The report consists of two sections with the following information:

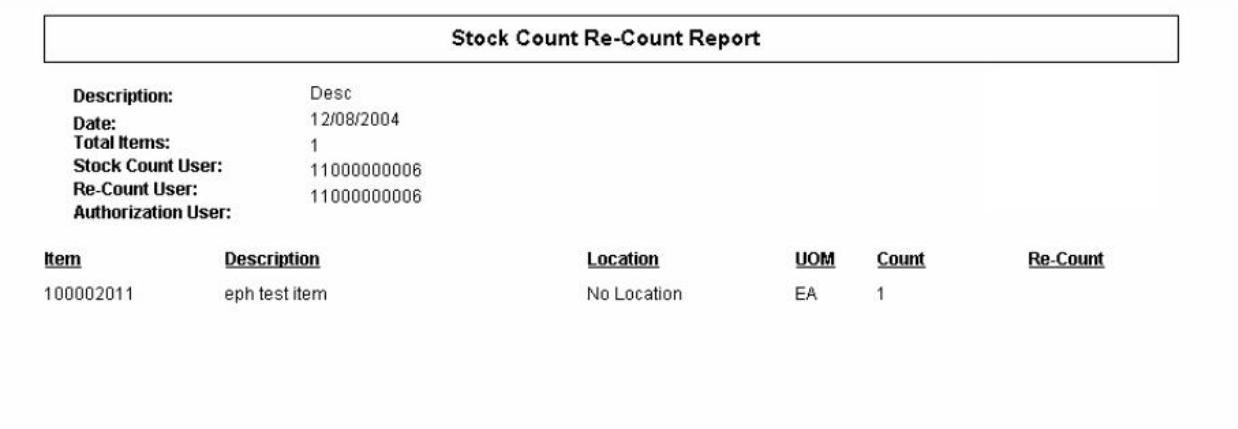
Header:

- Description – Stock count description
- Date – Scheduled date for the stock count
- Total Items – Total number of items in the stock count
- Stock count user – User who created the stock count
- Recount user – User who requested the recount
- Authorization user – User who authorized the stock count

Detail:

- Item – Item number for each line item in the stock count
- Description – Description of item
- Location – Sequenced location in the store being counted
- 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

Stock Count Re-count Report



Stock Count Re-Count Report

Description:	Desc
Date:	12/08/2004
Total Items:	1
Stock Count User:	11000000006
Re-Count User:	11000000006
Authorization User:	

Item	Description	Location	UOM	Count	Re-Count
100002011	eph test item	No Location	EA	1	

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 is based on the view STORE_ORDER_REPORT_V

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

Store Order Report

Store Order

Store Order Number: 8101

Store: 1000000026 -

Status: Pending

Supplier: 2345670000 - David Fashion Creations P/L

Creation Date: 22.03.2001

User:

Not Before Date: 27.06.2006

Not After Date: 28.06.2006

Comments:

Item	Description	Quantity(Units)	Unit Cost
100077195	Box of utensil	1	$1.00
100103445	nonsellable m&m pack	1	$1.00
100042048	Simple pack for 1000	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 is based on the view TRANSFER_REPORT_V

The report consists of two sections with the following information:

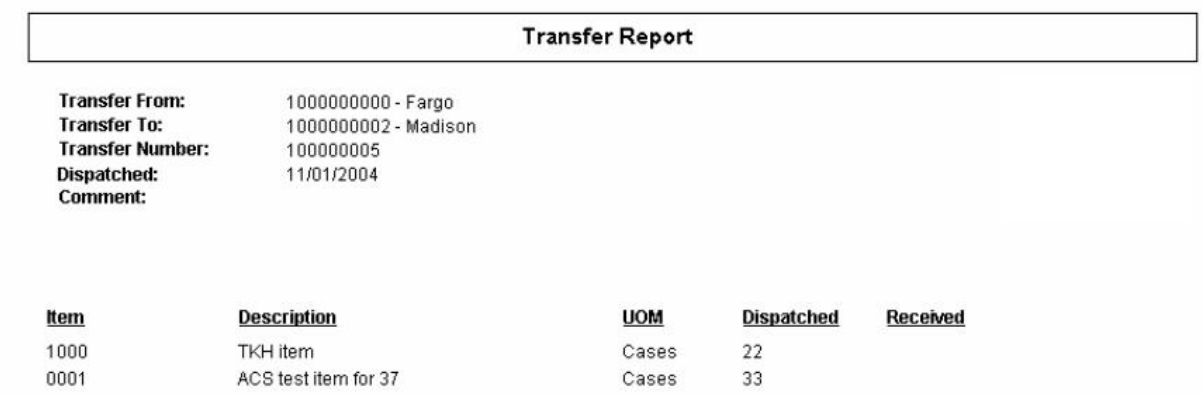
Header:

- 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

Transfer Report



Transfer Report

Transfer From: 1000000000 - Fargo
Transfer To: 1000000002 - Madison
Transfer Number: 100000005
Dispatched: 11/01/2004
Comment:

Item	Description	UOM	Dispatched	Received
1000	TKH item	Cases	22	
0001	ACS test item for 37	Cases	33	

6

Customization

Customization Overview

Retailers often modify retail software either in-house or have it modified through third-party system integrators.

If the customization efforts are not done correctly, the deployed product may 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.

This chapter should be used in conjunction with SIM 13.0.

Architecture

In order to ensure that the SIM application remains upgradeable for the client, all code-customization must follow the current architectural style.

- Do not connect directly to the RMS or SIM database using JDBC or similar code from the client or wavelink layer.
- The Wireless UI code base should never access the Swing UI code base, and vice versa.

Wireless User Interface

Guidelines for altering forms and developing in the wireless layer of code are contained within the Wireless Development section later in this chapter.

PC User Interface

Extending the Swing PC screens is difficult and requires code modifications. If code is modified, the custom.jar file must be placed first in the classpath.

Information about development and customization of the PC user interface layer can be found in the PC/User Interface Development section later in this chapter.

Server/Middle Tier

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 closed packages is considered essential to the integrity of the product and is not shared with retailers. Any modification of closed code will likely make the system unusable.

The code that resides in the shared packages is also critical to the functionality of the application, but does not have a clear or simple means of extension to produce custom functionality. Therefore, code may sometimes be modified to accommodate this functionality. All care should be given to read the appropriate documentation before modifying the code.

Modifying Business Objects

All business objects reside within the "closed" package structure. Information about business objects and development in the middle tier as well as customization can be found in the Business Layer Development section later in this chapter.

Modifying Services

All services reside within the "closed" package structure. Information about services and development in the middle tier as well as customization can be found in the Business Layer Development section later in this chapter.

Validation via Rules Engine

The Rules engine was designed to be configurable and therefore customizable. Information on how to develop rules can be found in the Business Layer Development section later in this chapter.

RIB/Injectors Related

Injectors should directly call DAOs and never call services.

If additional attributes or customization is needed in a RIB message, refer to RIB documentation.

To override a RIB injector to handle altered RIB messages, code a custom RIB injector and then reconfigure the RIB configuration file to point to the new custom injector.

Database Related

See the DAO Layer Development section for details on customizing and developing in the DAO layer.

Internationalization Related

The language translation of text within the application resides in translation tables on the database making modifications to the translations or English label text quite easy.

See the Internationalization section later in this chapter for further details on translating the SIM user interface display strings.

Build/Packaging/Deployment Related

Make a separate project for custom-modified code. Make sure any JAVA files that are going to be modified have the exact same class structure as 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 unsigning and resigning 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 unsigning/signing process.

Business Layer Development

Overview

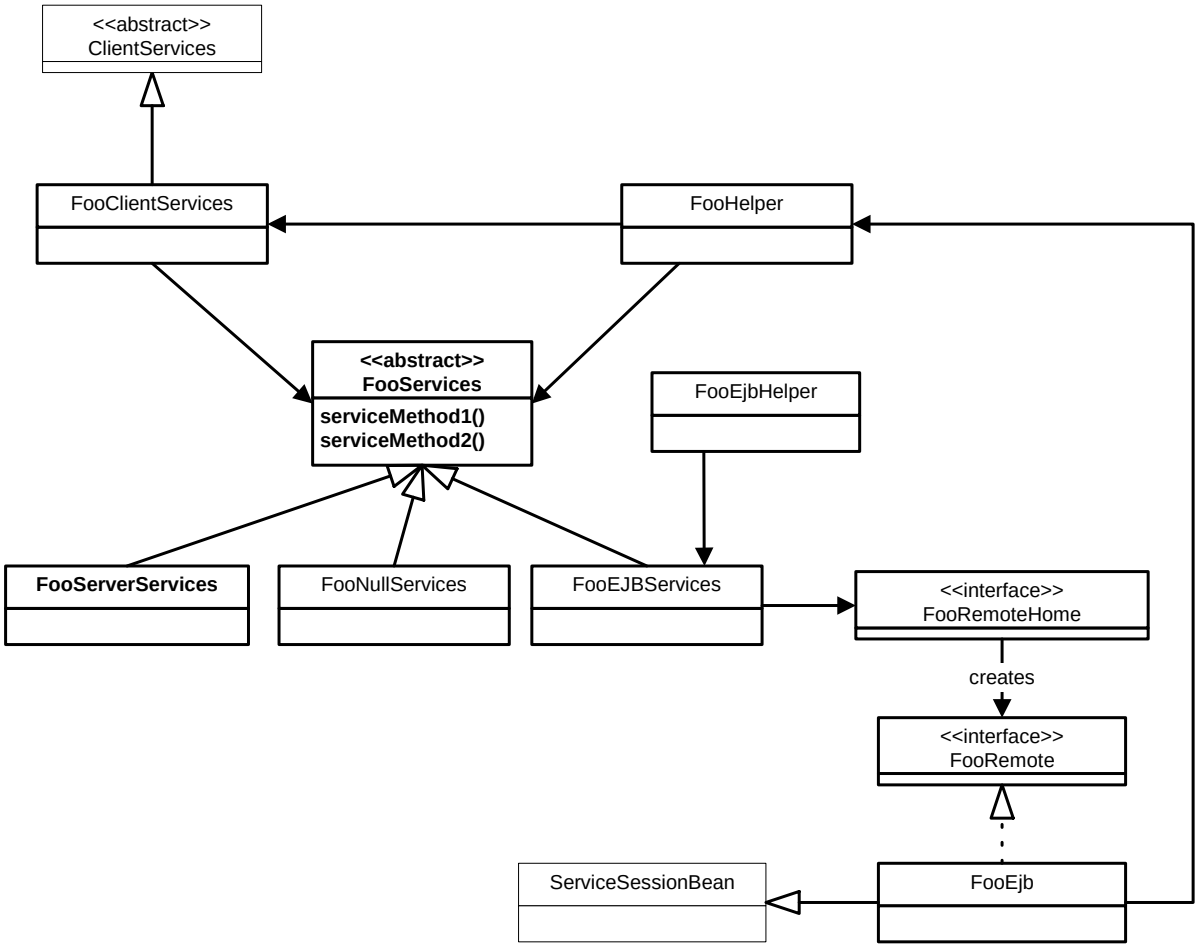
The business layer of the architecture consists primarily of services and business objects. Services execute on the server side of the SIM architecture and contain the majority of the business logic that takes place in the system. The server is designed to run in a single instance in a centralized corporate data center. Business objects capture concepts within the system, housing such ideas as a stock count and stock count line item.

Coding Guidelines

Always access services through the <servicename>Helper class, not through the <servicename>ClientServices or the <servicename>Services classes directly.

Services

Services are interfaces that define the relationship between the server and external clients. The interfaces are implemented as EJBs (Enterprise JavaBeans) in SIM and deployed into an EJB container (such as Oracle Application Server). A hierarchy of classes is created within SIM to make the usage of these services very simple. The Returns service will be used throughout the document to demonstrate server patterns.



<name>Services

This abstract class defines the services available on the server as well as any global constants used through the services. Since client/server interaction should be kept to a minimum, most services should be designed to handle large tasks such as retrieving full objects or full object collections and moving a business object through its life cycle. Fine-grained events such as adding a new line item to a return should be handled by business objects and rules and is not a good candidate for a service interface. See ReturnServices for an example of this class.

<name>Helper

By using this class to access the services, the calling class does not need to concern itself with retrieving references to EJB stubs or any of the underlying architecture. It does not even need to know whether it is executing on the client or on the server. The <name>Helper determines whether it is executing on the client or on the server and gets the appropriate service implementation. Each service method is defined as a static method call on the helper so that the developer may call the method directly.

<name>EjbHelper

Each service method is defined as a static method call on the helper so that the developer may call the method directly. When this class is used to access services, it ALWAYS accesses them by lookup and calling the EJB for the service. This class is intended to be used in code areas that are running on the server, but need to call services via EJB anyway. The only current place this is required is in server initialization classes (server bootstrap classes), but they may also be required for web service implementations in the future.

<name>ClientServices

This class is hidden beneath the <name>Helper so that the application developer does not need be concerned with this class. This class is specified in the SERVICES_LIST within client_master.cfg. Client startup automatically instantiates the classes within the cfg file and places a reference for usage within a global repository.

ClientServices

This is the abstract superclass for all <name>ClientServices classes. It provides methods to deal with going online and offline.

<name>NullServices

This class is a default offline implementation of the services. Each method returns null, false, an empty array, or an empty collection. If the server cannot be contacted, this implementation replaces the standard client implementation. This class is generated and hidden beneath “helper” classes so that the end developer does not need to concern him or her self with this class.

<name>EjbServices

This concrete class is executed on the client-side and executes the defined services by calling an Enterprise JavaBean implementation of the service. Each <name>EjbServices extends from <name>Services. This class is generated and hidden beneath “helper” classes so that the end developer does not need to concern him or her self with this class.

<name>ServerServices

This concrete class extends <name>Services and provides the actual implementation of the service functionality. This is the primary class that is coded by the application developer within the services structure. Services often instantiate one or more DAO classes to handle interaction with the persistence layer. Services typically do not instantiate and call other services, but if required, the second service should be called via its <name>Helper.

<name>Ejb

This concrete class is the server-side Enterprise JavaBean that executes the defined services. Each <name>Ejb extends from ServiceSessionBean, which extends from SessionBean. This class is hidden beneath “helper” classes so that the end developer does not need to concern him or her self with this class.

<name>RemoteHome

This is the remote home interface for the <name>Ejb Enterprise Java Bean. It is generated by XDoclet and used only in generated classes.

<name>Remote

This is the remote interface for the <name>Ejb Enterprise Java Bean. It is generated by XDoclet and used only in generated classes.

ServiceSessionBean

This is the abstract superclass for all <name>Ejb classes. It contains default implementations of the EJB lifecycle methods and holds onto the SessionContext instance for a <name>Ejb.

Writing Services

Creating an entirely new service indicates that a completely new feature of the system is being added to SIM or that customization is taking place. The following steps can be used to create or modify a service.

1. Create or modify the abstract base class for the service. This class is named <name>Service.

   Example: oracle.retail.sim.closed.activitylocking.ActivityLockingServices.

   ```
   public abstract class ActivityLockingServices {
   ```

2. Define a static attribute (often named current) along with a getCurrent() and setCurrent() method to retrieve and assign the attribute.

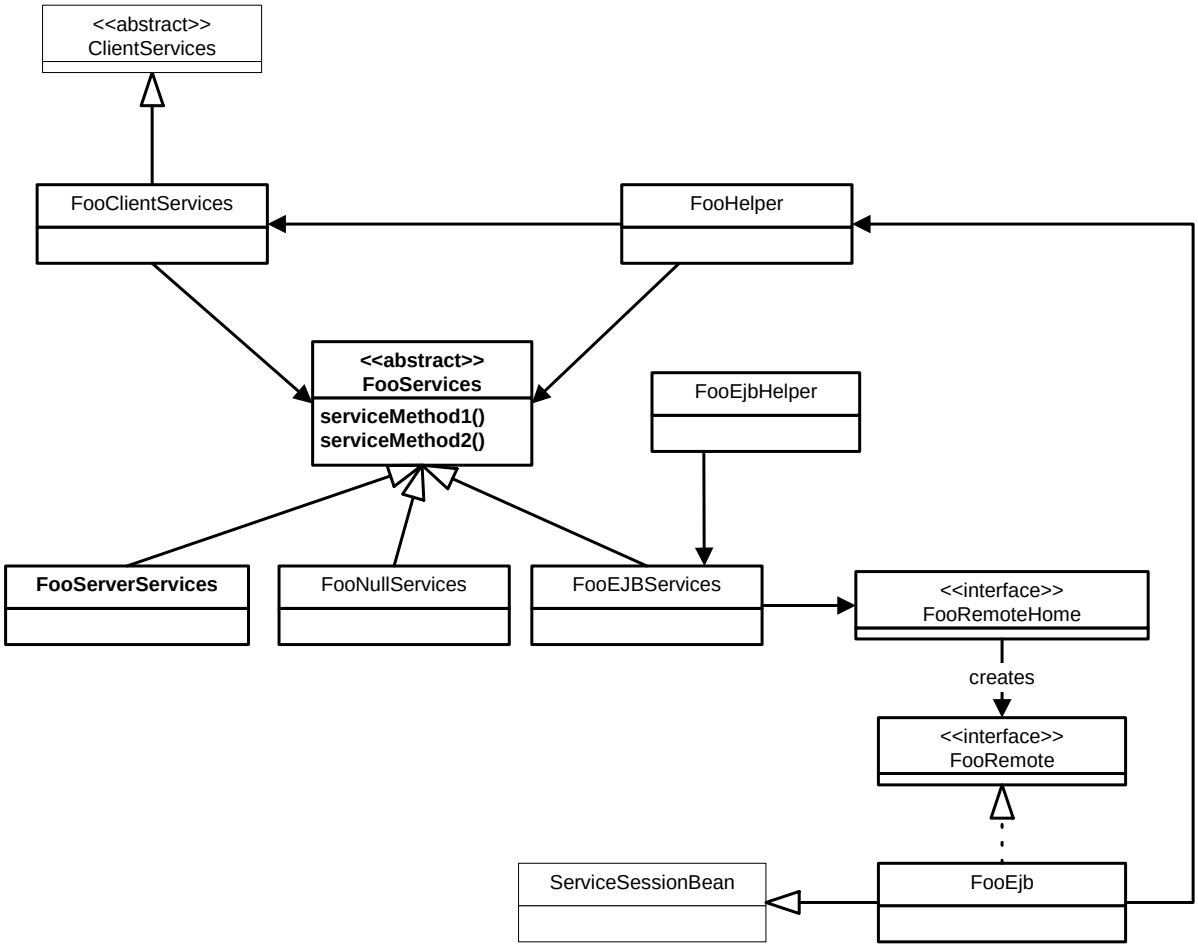
   ```
   private static ActivityLockingServices current = null;

   public static ActivityLockingServices getCurrent() {
       return current;
   }

   public static void setCurrent(ActivityLockingServices service) {
       current = service;
   }
   ```

3. Create the <name>ServerServices implementation of the services. This is a concrete class that implements the methods of the <name>Services abstract class.

   Example: oracle.retail.sim.closed.activitylocking.ActivityLockingServerServices.

   ```
   public class ActivityLockingServerServices extends ActivityLockingServices
   ```

4. Write all the various framework classes necessary to hook up client and server (see diagram). Understanding of EJBs and J2EE is necessary for this step. This includes:
   - *MyService*ClientServices
   - *MyService*Helper
   - *MyService*EjbHelper
   - *MyService*NullServices
   - *MyService*EJBServices
   - *MyService*RemoteHome
   - *MyService*Remote
   - *MyService*Ejb
   - *MyService*SessionBean


<<abstract>>
ClientServices
FooClientServices
FooHelper
<<abstract>>
FooServices
serviceMethod1()
serviceMethod2()
FooEjbHelper
FooServerServices
FooNullServices
FooEJBServices
<<interface>>
FooRemoteHome
creates
<<interface>>
FooRemote
ServiceSessionBean
FooEjb

Service Index

The following is a current list of service implementations in SIM Version 13.0.

- ActivityLockingServerServices
- AdhocCountAdminServerServices
- AuthServerServices
- BatchServerServices
- ConfigServerServices
- CustomThemeServerService
- DealServerServices
- EmployeeServerServices
- InventoryAdjustmentServerServices
- ItemServerServices
- ItemRequestServerServices
- ItemTicketServerServices
- MdseHierarchyServerServices
- PriceChangeServerServices
- ProcessMeasureAuditServerServices
- ProductGroupServerServices
- ProductGroupScheduleServerServices
- ReplenishmentServerServices
- ReportingServerServices
- ReturnServerServices
- SequencingServerServices
- ShipmentServerServices
- SourceServerServices
- StockCountServerServices
- StockCountLineItemServerServices
- StockCountLocationServerServices
- TransferServerServices
- StoreServerServices
- StoreOrderServerServices
- TranslationServerServices

Business Objects

Overview

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, TransferOut, TransferIn, 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 that use the business object.

Because they contain all the data, business objects flow across all three layers of the SIM architecture.

The Business Object Class

All SIM business objects extend from the single class BusinessObject. Many classes could be in a hierarchy chain, but BusinessObject should always be the top level object. Three main functions are provided by the class: rules execution, validation, and cloning.

- BusinessObject provides a checkForNullParameter() method. This can be used within all business object methods that do not allow a null value to be provided.
- 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 may not be what the developer wants. Sometimes, the logic requires that the business object be copied, but all objects referenced by the copy point to the same instances as the original business object. For the standard Java object implementation of clone(), use cloneShallow().
- It provides 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 may dynamically load and execute business rules defined for a method.

Developing Business Objects

1. The new business object must directly or indirectly extend BusinessObject. This provides the generic functionality of the business object. Vendor Return is an example of an indirectly extended Business Object. Class declarations are included here to illustrate.

   ```
   public abstract class VendorReturn extends Return

   public abstract class Return extends BusinessObject
   ```

2. Unless the attribute is read-only, define a 'set<attribute>' method. This method is used to set the attribute value on the object.
   - The setter signature must be defined to allow BusinessException to be thrown.
   - If the attribute cannot be null, insert code to validate for the null parameter.
   - Always insert code to execute a rule for the attribute.
   - Always call doSet<attribute> to finally assign the value.

   ```
   public void setStatus(Integer status) throws BusinessException {
       checkForNullParameter("setStatus", status);
       executeRule("setStatus", status);
       doSetStatus(status);
   }
   ```

3. Define a 'doSet<attribute>' method. It is this method that actually assigns the value to the business object. Usually doSet<attribute> is only called in the DAO layer or by the 'set' method for the same attribute. Calling the doSet<attribute> from any other code is highly discouraged as this leads to incoherent objects that can break the system.

```
public void doSetStatus(Integer status) {
  this.status = status;
}
```

4. If a retailer requires the ability to determine whether or not certain attributes are modifiable or not based on the current state of the object, implement the isPropertyModifiable(String attribute) method. This method makes an executeRule() call just like many setters do, but it 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 isPropertyModifiable business rule is just like that of any other business rule. It makes whatever checks are necessary to determine if the specified attribute is modifiable at this time. It returns a RulesInfo object containing a text message if it is NOT modifiable. Otherwise it returns an empty RulesInfo object.

```
public boolean isPropertyModifiableNew(String propName) {
  try {
    executeRule("isPropertyModifiable", new Object[] { propName, this });
  } catch (BusinessException bre) {
    return false;
  }
  return true;
}
```

Persisting Business Objects

Business objects are persisted through the DAO layer of the architecture. A new DAO must be created or an existing DAO altered in order to persist the business object. See the DAO Layer Development section later in this chapter for details on this process.

Creating a New 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. Here are the steps for create a new QueryFilter.

1. Create a filter <name>QueryFilter that implements the QueryFilter interface that also implements Serializable.

   For example: StockReturnQueryFilter

```
public class StockReturnQueryFilter implements QueryFilter {
    static final long serialVersionUID = -7878416835903774879L;
    static final int TWO_WEEKS = -14;

    private String itemId = null;
    private String returnId = null;
    private Integer status = null;
    private String userId = null;
    private String supplierId = null;
    private String warehouseId = null;
    private String authCode = null;
    private String reason = null;
    private Date fromDate = null;
```

```
    private Date toDate = null;
    private String storeId = null;

    public StockReturnQueryFilter() {
        super();
    }
```

2. If the QueryFilter class should have business logic rule validation enabled, then the class also needs to extend BusinessObject.

   ```
   public class StockReturnQueryFilter extends BusinessObject implements
   QueryFilter
   ```

3. Define the attributes of the filter. Most of the values will be attributes that can be used within a WHERE clause to retrieve the business domain object (criteria of the business object). Sometimes, a flag or other unique value will not map directly to the business object, but this very rare.
4. Define the accessor methods of the filter class. The style of getters and setters that should be used is defined by whether or not BusinessObject has been extended.
5. The query filter is ready to use. Services and DAO layer implementations must be altered to use the new query filter.

Commands

Commands are used inside service methods to keep complex and unique code separated from the service class. At the moment, this is primarily used as an organization technique.

Creating a New Command

1. All commands must subclass Command and the class name must end with the word Command.

   ```
   public class DirectDeliveryUpdateCommand extends Command
   ```

2. Create constructor or setters for any properties that need to be set on the command for execution. From a server service, this will often be DAO classes or business objects.

   ```
   public DirectDeliveryUpdateCommand(ShipmentDao shipmentDao, PurchaseOrderDao
   purchaseOrderDao) {
       this.shipmentDao = shipmentDao;
       this.purchaseOrderDao = purchaseOrderDao;
   }

   public void setDirectDelivery (DirectDelivery delivery) {
       this.delivery = delivery;
   }
   ```

3. Create getters for information that needs to be retrieved when the command is finished executing.

   ```
   public DirectDelivery getDirectDelivery() {
       return delivery;
   }
   ```

4. By subclassing Command, you will be forced to implement doExecute(), the abstract method in the superclass. The logic that the command should perform all goes here, though private helper methods are encouraged to keep the code clean.

   See DirectDeliveryUpdateCommand for complete method.

   ```
   protected void doExecute() throws Exception {
     // Place your code here
   }
   ```

5. Use the newly created command in your ServerService code. Instantiate it, set attributes, execute and retrieve information.

   This example is a service in ShipmentServerService

```
public DirectDelivery updateDirectDelivery(DirectDelivery delivery) throws
Exception {
DirectDeliveryUpdateCommand command = new
DirectDeliveryUpdateCommand(shipmentDao, purchaseOrderDao);
command.setDirectDelivery(delivery);
command.execute();
return command.getDirectDelivery ();
}
```

6. If a service needs to be accessed, use the <name>Helper class to access the service so that the appropriate lookup of the service takes place. Note that rules are executed on the client with great frequency. Making a service call within the rule has a negative performance impact and should be avoided if at all possible.

   Note: The command is guaranteed to be running on the server.

Rules

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 (see the executeRule() method on BusinessObject). 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 oracle.retail.sim.closed.rules.* packages. If a desired rule does not exist, follow the steps below to create a new rule.

Creating a New Business Rule

1. All rules must subclass SimRule.

```
public class ItemMustBeRangedRule extends SimRule {
```

2. Override the execute() method. This first method may do 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 actually logic of the business rule.

   In the example below, the object parameter passed in the by the rule engine is not needed. The first parameter of the array is the StockItem that needs to be validated.

```
public RulesInfo execute(Object object, Object[] args) {
  return execute((StockItem) args[0]);
}
```

3. Implement the typed execute() method with the actual logic necessary to perform the desired validation.

```
public RulesInfo execute(StockItem stockItem) {
try {
    // If there is no stockable, we don't want the rule to fail
    if ((stockItem!= null) && (stockItem.getId() != null)) {
        String storeId = UniversalContext.getStoreId();
        if (!ItemHelper.isRanged(stockItem.getId(), storeId)) {
            return RULE_FAILED;
        }
    }
    return RULE_PASSED;
} catch (Exception ex) {
    log(ex);
```

```
}
return RULE_FAILED;
    }
```

- In the above 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 SimRule that should be used in all subclasses at the appropriate spots.
- SimRule provides a few log() methods to log exceptions that occur within the validation. It is standard practice to catch and log errors and then return RULE_FAILED.
- If a service needs to be accessed, then the above example is the approach to follow. Use the <name>Helper method to access the service so that the appropriate lookup of the service takes place. Note that rules are executed on the client with great frequency. Making a service call within the rule has a negative performance impact and should be avoided if at all possible.
- 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 will likely leave the business object in a non-coherent state.

4. Update the \files\prod\config\retek\rules_sim.xml file.

```
<object className="oracle.retail.sim.closed.pricechange.PriceChange">
  <property id="isCoherent" propertyNames="isCoherent">
     <rule_class
className="oracle.retail.sim.closed.rules.pricechange.PriceChangeIsCoherentRul
e"/>
  </property>
  <property id="isPropertyModifiable" propertyNames="isPropertyModifiable">
    <rule_class className=
"oracle.retail.sim.closed.rules.pricechange.ArePriceChangePropertiesModifiable
Rule"/>
  </property>
  <property id="setEffectiveDate" propertyNames="setEffectiveDate">
    <rule_class
className="oracle.retail.sim.closed.rules.common.DateAfterTodayRule"/>
  </property>
  <property id="setNewPrice" propertyNames="setNewPrice">
    <rule_class
className="oracle.retail.sim.closed.rules.common.CurrencyMustBePositiveRule"/>
  </property>
  <property id="setStockable" propertyNames="setStockable">
    <rule_class
className="oracle.retail.sim.closed.rules.common.ItemMustBeRangedRule"/>
    <rule_class
className="oracle.retail.sim.closed.rules.common.ItemMustBeSellableRule"/>
    <rule_class
className="oracle.retail.sim.closed.rules.common.ItemPriceMustBeStoreControlle
dRule"/>
  </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.pricechange.PriceChange">
</object>
```

Within the class definition is a property definition where the id is the method signature on which the validation should be done.

```
<property id="setStockItem" propertyNames="setStockItem">
</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.ItemMustBeRangedRule"/>
```

Value Objects

A value object (VO) is a trimmed down version of a business object done to reduce data flow in areas that do not require attribute updating or business logic.

Creating a VO

1. Determine the business object and properties that the VO is to be a version of and create the VO class in the same package with the same name – ending in the letters VO. For example, Supplier becomes SupplierVO. All VOs should be serializable.

   ```
   public abstract class SupplierVO implements Serializable {
   ```

2. Declare as private class variables ONLY those values that will be directly used by the VO for its limited scope of usage. SupplierVO is used in supplier lookup functionality, so only rin (id) and name are needed.

   ```
   private String rin = "";
   private String name = "";
   ```

3. Declare constructors for the VO. If the number of attributes of the object is small enough, then a constructor should be declared passing in the parameters.

   ```
   public SupplierVO(String rin, String name) {
     this.rin = rin;
     this.name = name;
   }
   ```

4. Declare getters for the values on the VO.

   ```
   public String getRin() {
     return rin;
   }

   public String getName() {
     return name;
   }
   ```

5. VOs never have setters(). VOs are read-only objects that are never modifiable.
6. If there are too many attributes for a convenient constructor (five or more attributes is a good rule of thumb), then doSet() methods should be created for each attribute. These methods should never do validation or any other logic, but simply assign the value. They should only be used by the DAO layer to populate the VO.

Server Initialization Classes

Server Initialization Classes are defined as classes that implement the oracle.retail.sim.closed.common.Initializer interface and are entered in the INITIALIZE list in the server_master.cfg file.

When the server is started, the ServerBootstrap is notified, reads the server_master.cfg file to find the list of classes for the INITIALIZE key, and instantiates and executes each of those classes.

If you need to perform some process on server startup, create a new class that implements oracle.retail.sim.closed.common.Initializer:

```
public class TestInitializer implements Initializer {
```

```
  public void executeInitialization() throws Exception {
    ReturnQueryFilter filter = BOFactory.createStockReturnQueryFilter();

    List returnVos = ReturnEjbHelper.findReturnVOs(filter);
    LogService.info(this, "Query found " + returnVos.size() + " returns.");
  }
}
```

Enter the name of your class in the INITIALIZER list in server_master.cfg:

```
# This file contains information important to the server

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

Access to Services

Server Initialization classes do not have access to normal server resources – for instance, direct calls to DAO methods will not work. If they need to do work that accesses the database, they need to call a service to do that work.

Calling services from a server initialization class is different than calling a service normally. Server initialization classes MUST use the **<name>EjbHelper** class to call a service, or the service call will fail.

Key Classes

The following list is the key classes within the business layer. Those working in the business layer should take the time to familiarize themselves with these classes in their entirety.

- BackgroundRunner
- BusinessObject
- BusinessException
- ServiceSessionBean
- Command
- QueryFilter
- RulesInfo
- SimConfig
- SimRule
- SimRuleBusinessProblem

Customizing the Business Layer

This section covers tips on customizing the service layer.

Customizing a Business Object

> **Note:** Do not change business objects.

In order to change the behavior or add additional attributes to a business object, follow the process below.

1. Create a new object that subclasses the original business object. Add new attributes, getters, setters or other functional methods to this new class. It is also possible to override public or protected methods of its parent class.

   Example:

   ```
   public class MySupplier extends Supplier {
   ```

2. Customize the BOFactory to instantiate the new object. See Customizing The BOFactory. Because all SIM business objects are instantiated with the BOFactory, all services and code that used to return Supplier business objects now return MySupplier business objects.

   Example: readSupplier() service used in handheld or pc code

   ```
   Supplier supplier = SourceHelper.readSupplier(supplierId);

   can now become

   MySupplier supplier = (MySupplier) SourceHelper.readSupplier(supplierId);
   ```

3. Use the new object within your code.

> **Note:** If you have added attributes to DB tables, you need to customize the DAO layer.

Customizing the BOFactory

The BOFactory is used to instantiate all business objects. The implementation of the business object factory is determined by the "bofactory.cfg" file. These are the steps to follow to implement a customized BO Factory.

1. Create a class that extends BOFactoryImpl.

   Example:

   ```
   public class MyBOFactoryImpl extends BOFactoryImpl {
   ```

2. Alter the setting in the configuration file to use the new class.

   Example: bofactory.cfg

   ```
   # This class will be used to instantiate new business objects in SIM.
   FACTORY_IMPL=my.custom.classpath.MyBOFactoryImpl
   ```

3. Override methods to return the customized class.

   Example: MyBOFactoryImpl

   ```
   public Supplier createSupplier() {
     return new MySupplier();
   }
   ```

Creating a New Service

Follow the steps for writing a service.

Customizing an Existing Service

Note: Do not change services.

In order to override the method on an existing service to add or alter functionality, follow the process below.

1. Create a class that extends the server services.

   Example:

   ```
   public class MySourceServerServices extends SourceServerServices {
   ```

2. Alter the setting in the configuration file to use the new class.

   Example: services.cfg

   ```
   SourceServices.CLIENT_IMPL=oracle.retail.sim.closed.source.SourceEJBServices
   SourceServices.CLIENT_DOWNTIME=oracle.retail.sim.closed.source.SourceNullServi
   ces
   SourceServices.SERVER_IMPL=my.custom.classpath.MySourceServerServices
   ```

3. Override methods to implement your own services.

   Example: MySourceServerServices

   ```
   public Supplier readSupplier(String id) throws Exception {
     Supplier supplier = supplierDao.selectSupplier(id);
     // Custom code to manipulate supplier
     return supplier;
   }

   // OR
   public List<SupplierVO> findSupplierVOs(SourceQueryFilter filter)
                                 throws Exception {
     List<SupplierVO> vos = super.findSupplierVOs(filter);
     // Custom code to manipulate supplier vos
     return vos;
   }
   ```

DAO Layer Development

Altering the DAO Layer

Once a business object is created or altered, you need to modify the DAO layer to persist and retrieve the information properly.

1. Update the database tables to contain the new information (see "Database Tables", below).
2. Alter the DAO interface and implementation to handle persistence or retrieval of the new information (see "Developing DAO Classes").

Database Tables

There are two standard categories of database tables in SIM: ARTS tables and SIM specific tables. ARTS tables are based on an industry standard definition. It may be helpful to know that to decode the names of an ARTS table, it helps to read the table backwards. For example, the PA_EM table represents Employee Party. SIM specific tables all begin with the **RK_** prefix to indicate that they are not parts of the ARTS model.

DAO Configuration

The DAO configuration file contains a KEY used within the code mapped to the full pathname to the class that should be instantiated and executed. If creating a new DAO, add an entry to the DAO config file.

```
PURGE_DAO=oracle.retail.sim.shared.dataaccess.sim.oracle.dao.PurgeOracleDao
DEALS_DAO=oracle.retail.sim.shared.dataaccess.rsl.DealsRSLDAO
```

The DAO configuration file is used by the DaoConfigManager class to retrieve the configured dao implementation class.

Example: ItemRequestServerServices

```
private ProductGroupScheduleDao scheduleDao = null;
private ItemRequestDao itemRequestDao = null;
private ProcessMeasureAuditDao auditDao = null;

public ItemRequestServerServices() {
  scheduleDao = (ProductGroupScheduleDao)
      DaoConfigManager.getDao(ProductGroupScheduleDao.CONFIG_KEY);
  itemRequestDao = (ItemRequestDao)
      DaoConfigManager.getDao(ItemRequestDao.CONFIG_KEY);
  auditDao = (ProcessMeasureAuditDao)
      DaoConfigManager.getDao(ProcessMeasureAuditDao.CONFIG_KEY);
}
```

Developing DAO Classes

This section outlines general design and patterns for working with code within the DAO layer of the system.

Create DAO Interface

When customizing the database, new DAO classes should always be created to handle the custom code. To begin, a DAO interface is created that defines what the DAO is responsible for. An interface is used within the code and the actual implementation of the DAO interface is determined a runtime. This allows customization of the DAO by swapping out the implementation at a client site. Below is a good demonstration of a DAO interface.

```
/**
 * Interface for Item Request DAO Objects.
 * Copyright © 2004, 2007, Oracle. All rights reserved.
 */
public interface ItemRequestDao {
    public static final String CONFIG_KEY = "ITEM_REQUEST_DAO";

  /**
   * Persist the item request.
   * @param itemRequest The item request to be persisted.
   */
  public void insert(ItemRequest itemRequest) throws DAOException;

  /**
   * Update the item request.
   * @param itemRequest The item request to be updated and persisted.
   */
  public void update(ItemRequest itemRequest) throws DAOException;

  /**
   * Locate the item request by its id.
   * @param requestId The item request ID.
   * @return The item request or null if none is found.
   */
  public ItemRequest selectItemRequest(String requestId) throws DAOException;

  /**
   * Locate the item request by the criteria set on the filter.
   * @param filter Filter representing the search criteria to match.
   * @return List of item requests meeting the search critera, empty if none
   *     are found.
   */
  public List<ItemRequest> selectItemRequests(ItemRequestQueryFilter filter)
throws DAOException;

  /**
   * Delete the item requests created by the batch today and not yet processed.
   */
  public void deleteItemRequestsForBatch() throws DAOException;
}
```

Some basic principals of DAO interfaces:

- A DAO interface does not extend any classes. It is a stand alone class. It should be named <*name*>Dao.
- The first value in the class is the CONFIG_KEY, which contains the key used in the dao.cfg file to look up the actual implementation. This should always be present.

Create or Update DAO Implementation

Once the interface has been altered, the base implementation needs to be changed as well.

```
public class ItemRequestOracleDao extends BaseOracleDao implements ItemRequestDao
```

- The class declaration of the implementation should be named <*name*>OracleDao. The classpath to this class needs to exist in the dao.cfg file for the CONFIG_KEY found in the interface.
- The DAO implementation should always extend BaseOracleDao, which supplies many of the common functions required at this layer.
- The DAO implementation implements the interface.

```
public void insert(ItemRequest itemRequest) throws DAOException {
  if (!ItemRequest.NEW_ITEM_REQUEST_ID.equals(itemRequest.getId())) {
    throw new DAOException("unable to insert item request");
  }
  itemRequest.doSetId(getNextItemRequestId());
  execute(getInsertSql(itemRequest));
}

private List<ParametricStatement> getInsertSql(ItemRequest itemRequest)
                            throws DAOException {
   List<ParametricStatement> statements = new ArrayList();
   List params = fromObjectToBean(itemRequest).toList();
   statements.add(new ParametricStatement(RkItemRequestDataBean.INSERT_SQL,
params));
  // Also insert all the line items for the item request
  RkItemRequestLineItemDataBean bean = new RkItemRequestLineItemDataBean();
  bean.setItemRequestId(itemRequest.getId());

  for (ItemRequestLineItem lineItem : itemRequest.getLineItems()) {
    bean.setItemId(lineItem.getStockItem().getId());
    bean.setQuantity(lineItem.getQuantity().doubleValue());
    bean.setPackSize(lineItem.getCaseSize().doubleValue());
    lineItem.doSetClean();
    lineItem.doSetPersisted();
    statements.add(
      new ParametricStatement(RkItemRequestLineItemDataBean.INSERT_SQL,
      bean.toList()));
   }
  return statements;
}
```

The above code shows some basic insert SQL at the DAO layer. Helper methods were created in this class to instantiate a ParametricStatement object around the SQL and the list of parameters. The parameter statement is passed to the execute() method found in the superclass BaseOracleBean.

```
public ItemRequest selectItemRequest(String requestId) throws DAOException {
    if (requestId == null) {
        return null;
    }

    // do a select on id
```

```
    String sql = RkItemRequestDataBean.SELECT_SQL +
        where(RkItemRequestDataBean.COL_ITEM_REQUEST_ID);
    List params = new ArrayList();
    params.add(requestId);

    RkItemRequestDataBean[] beans = (RkItemRequestDataBean[]) query(new
        RkItemRequestDataBean(), sql, params);
    if (!hasBeans(beans)) {
        return null;
    }

    itemRequest = fromBeanToObject(beans[0]);

    // Now attach merchandising hierarchies and single items to it:
    addLineItems(itemRequest);

    return itemRequest;
}
```

This is a basic find method to retrieve data by an ID. The SQL and parameters are created and then the query() method is called on the superclass BaseOracleBean. This retrieves an array of the data beans that represent the retrieved data. After checking whether or not beans were successfully retrieved, the bean is converted to the business object in the method fromBeanToObject().

```
private ItemRequest fromBeanToObject(RkItemRequestDataBean bean) throws
  DAOException {
    StoreDao storeDao = new StoreOracleDao();
    ItemRequest itemRequest =
     BOFactory.createItemRequest(storeDao.selectStore(bean.getStoreId()));
    itemRequest.doSetId(bean.getItemRequestId());
    itemRequest.doSetCreateDate(bean.getCreateDatetime());
    itemRequest.doSetDepartmentId(bean.getDept());
    itemRequest.doSetExpirationDate(bean.getExpiryDate());
    itemRequest.doSetReqDeliveryDate(bean.getReqDeliveryDate());
    itemRequest.doSetStatus(new Integer(bean.getStatus().intValue()));
    itemRequest.doSetEmployeeId(bean.getUserId());
    itemRequest.doSetProcessDate(bean.getProcessDate());
    itemRequest.doSetBatchCreated(bean.getCreatedBy().equals(BATCH));
    itemRequest.doSetComments(bean.getComments());
    itemRequest.doSetScheduleDesc(bean.getScheduleDescription());

    return itemRequest;
}
```

Included here is an example of converting a bean to an object. Note that doSet<*attribute*>() methods should be used when converting bean data to the business object. The regular set() methods trigger business logic that does not apply when reading data from the database.

Stored Procedures

This section covers the design patterns/steps when using a stored procedure (for example, CallableStatement) in the DAO layer.

1. Create a class that implements SimStoredProcedure

   Example: GenerateItemRequestProcedure

   ```
   public class GenerateItemRequestsProcedure implements SimStoredProcedure {
   ```

2. Implement the getSql() method to return the callable statement SQL.

   ```
   public String getSql() {
       return "call GENERATE_ITEM_REQUESTS.GET_STOCK_COUNT (?,?,?,?)";
   }
   ```

3. Implement the registerParameters() method. Register the types and sequence of the input and output parameters of the callable statement.

```
public void registerParameters(CallableStatement statement)
throws SQLException {
    SimpleDateFormat dateFormatter = new SimpleDateFormat("MM/dd/yyyy");
    statement.registerOutParameter(1, Types.VARCHAR);
    statement.registerOutParameter(2, Types.VARCHAR);
    statement.setString(3, dateFormatter.format(new Date()));
    statement.registerOutParameter(4, Types.VARCHAR);
}
```

4. Implement the processResults() method to retrieve the desired output from the callable statement.

```
public void processResults(CallableStatement statement) throws SQLException {
    errorMessage = statement.getString(2);
}
```

5. Implement the error handling methods.

```
public boolean hasError() {
    return ((errorMessage != null) && (errorMessage.trim().length() > 0));
}

public String getError() {
    return errorMessage;
}
```

6. Use the new class in the DAO layer. executeStoredProcedure() is a BaseOracleDao method that handles the execution of a stored procedure through the SimStoredProcedure interface.

```
public void generateItemRequests() throws DAOException {
    GenerateItemRequestsProcedure procedure
       = new GenerateItemRequestsProcedure();
    executeStoredProcedure(procedure);
    if (procedure.hasError()) {
      throw new DAOException("Error from " + procedure.getError());
    }
}
```

7. If you need to pass input parameters or get output parameters, use the constructor or public methods.

   Example: GenerateUnitStockCountProcedure (see class for details of parameters usage)

```
private Date date = null;
private long stockCountId = -1;
private long numberOfRecords = 0;

public GenerateUnitStockCountProcedure(Date date, long stockCountId) {
    this.date = date;
    this.stockCountId = stockCountId;
}

public long getNumberOfRecords() {
    return numberOfRecords;
}
```

Databeans

Databeans are manipulated by the framework to insert, update and remove information from the database. Each bean wraps some amount of SQL and the parameters necessary to execute it. There are four types of databeans that may need to be created at the DAO layer: generated basic, generated custom, generated select and custom.

Basic Databean: A basic databean is one that maps all columns to and from a database table and incorporates all the necessary SQL.

Select Databean: A select databean is one that maps only some of the columns from a database table and incorporates all the necessary SQL to select only the information. This type of bean does not allow inserts or updates of the information.

Custom Databean: A fully custom databean is one that has very complex SQL, connects to multiple tables, or in some way is not a singular mapping of Java code to the database. Rather than building large chunks of SQL within a DAO method, Oracle Retail suggests designing a custom bean that represents the SQL (a SQL wrapper of sorts). To do so, follow or copy a pre-existing custom databean and modify the code by hand so that the bean represents exactly what you want.

Key Classes

The following classes are key superclasses in the DAO layer:

- BaseOracleBean
- BaseOracleDao
- DAOException
- DatabaseNull
- DataSourceDbConnectionFactory
- DbConnectionFactory
- ParametricStatement
- SimStoredProcedure

Customizing the DAO Layer

This section contains a few tips about customizing the DAO layer of the application.

Database Related Tips

- Removing a column from the database will break the code that attempts to read that table. Do not remove columns!
- New tables should always begin with a custom client created prefix. Example: ***MY***_TABLE_NAME.
- New columns added to a database table should begin with a custom prefix. Example: ***MY***_COLUMN_NAME
- All new columns added to the existing tables need to 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.

Creating New DAO

- Create new DAO interface and DAO implementation classes as described in the document above.
- Remove the dao.cfg properties file from its JAR, alter it to include the new DAO interface, and put it back in the JAR.

Extend Current DAO

A good technique to add features to an already existing DAO is to extend the class and then reconfigure the DAO.cfg to point to it. To extend a DAO, simply pick the appropriate type of DAO, such as ItemDao for item information, and create a class that extends it. In this manner, you will have access to all the available public methods, be able to override methods without altering the original file, and add additional methods.

Example: Extending DAO

```
public class MyCompanyItemOracleDao extends ItemOracleDao {
```

> **Note:** Be sure to reconfigure the DAO.cfg file to point ITEM_DAO to the new class.

Example: Dao.cfg

```
ITEM_DAO=my.classpath.in.custom.code.MyCompanyItemOracleDao
DEALS_DAO=oracle.retail.sim.shared.dataaccess.rsl.DealsRSLDAO
```

Additional Attributes

If an additional attribute of some data is required, columns may be added onto the end of a table. If this is the case, custom SQL or custom bean classes are required to read the information from the database. Current beans should continue to work as long as the column is nullable.

It is also suggested that a completely new DAO interface and implementing class be written to accommodate the usage of the new code. Altering existing DAO implementation classes will break future releases of the code or at the least, make upgrading difficult.

Exceptions and Logging

Exceptions designed for SIM are described in the first section, followed by usage examples.

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 values is reached and then it resets to (1) again. The ID is used to uniquely identify an exception within a server log. This class may be instantiated around another exception, such as a SQLException.

Downtime Exception

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 will be handled by the EJB before propagating the exception to the client. The database framework generates most SimServerException, though sometimes logic requires manually throwing a SimServerException. Below is an example of a DAO interface declaration and throwing a SimServerException manually.

Example: ItemDao

```
public List<StockItem> selectStockItems(StockItemQueryFilter filter, String
storeId) throws SimServerException;
```

Example: ProductGroupOracleDao

```
public void insert(ProductGroup group) throws SimServerException {
    if (group.getId() != null) {
        throw new SimServerException("Unable to insert product group!");
    }
    group.doSetId(getNextGroupId());
    execute(new ParametricStatement(RkProductGroupDataBean.INSERT_SQL,
         fromObjectToBean(group).toList()));
    insertProductGroupDetails(group);
}
```

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. Below is an example of a service declaration and throwing a BusinessException 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: ReplenishmentServices

```
public PickList updatePickList(PickList pickList) throws Exception {
    PickListUpdateCommand command = new PickListUpdateCommand(pickListDao,
itemDao);
    command.setPickList(pickList);
    command.execute();
    return command.getPickList();
}
```

Example: PickListUpdateCommand

```
protected void doExecute() throws Exception {
    PickList dbPickList = pickListDao.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);
    }
    if (dbPickList.getStatus() == PickListStatus.PENDING &&
           pickList.getStatus() == PickListStatus.COMPLETE) {
        for (PickListLineItem lineItem : pickList.getLineItems()) {
            lineItem.doSetActualPickAmount(lineItem.getPickAmount());
        }
    }
    . . . . . .
}
```

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, simply throw a BusinessException with the appropriate message. The exception should almost always be propagated to the last place that handled an event in order to cleanly break the execution flow.

Example: ItemTicketListModel

```
public void updateStockOnHand(ItemTicket itemTicket) throws Exception {
    Quantity stockOnHand = itemTicket.getItem().getAvailableStockOnHand();
```

```
    if (stockOnHand.intValue() < 1) {
        throw new BusinessException(ItemTicketConstants.ITEM_NOT_UPDATED);
    }

    itemTicket.setQuantity(stockOnHand.intValue());

    if (!obtainLock(itemTicket.getId(), ActivityLockingType.ITEM_TICKET)) {
        throw new BusinessException(ItemTicketConstants.LOCK_TAKEN_OVER);
    }

    ItemTicketHelper.updateItemTicket(itemTicket);
    ActivityLockingUtility.releaseLock(itemTicket.getId(),
        ActivityLockingType.ITEM_TICKET);
}
```

Catching an Exception

Exceptions should almost 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.

Example: ItemTicketListScreen

```
    public void navigationEvent(NavigationEvent event) {
        String command = event.getCommand();
        try {
            if (command.equals(SimNavigation.DONE)) {
                panel.handleDone();
            } else 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);
        }
    }
```

The alternate case to displaying an exception where an event is handled is when code must execute 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.

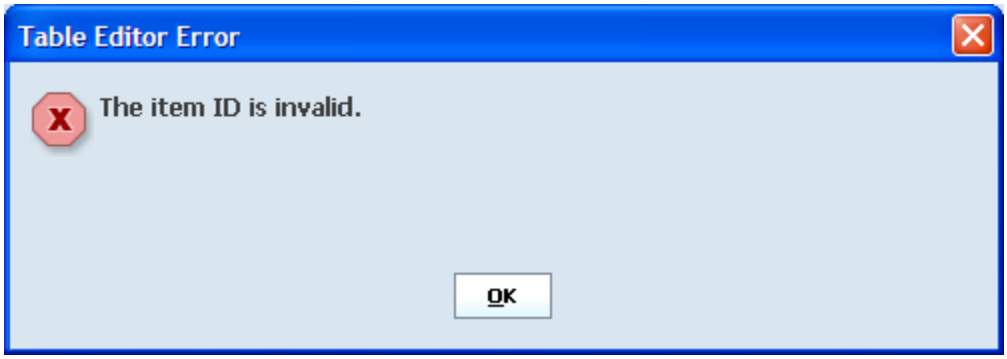
Example: ItemTicketListPanel

```
public void handleUpdateStockOnHand() throws Exception {
    if (itemTicketTable.getSelectedRowCount() == 0) {
        displayError(ItemTicketConstants.NO_ROWS_FOR_UPDATE);
        return;
    }

    { . . . . }

    ItemTicket itemTicket = null;
    for (Iterator iterator =
       itemTicketTable.getAllSelectedRowData().iterator(); iterator.hasNext();)
    {
```

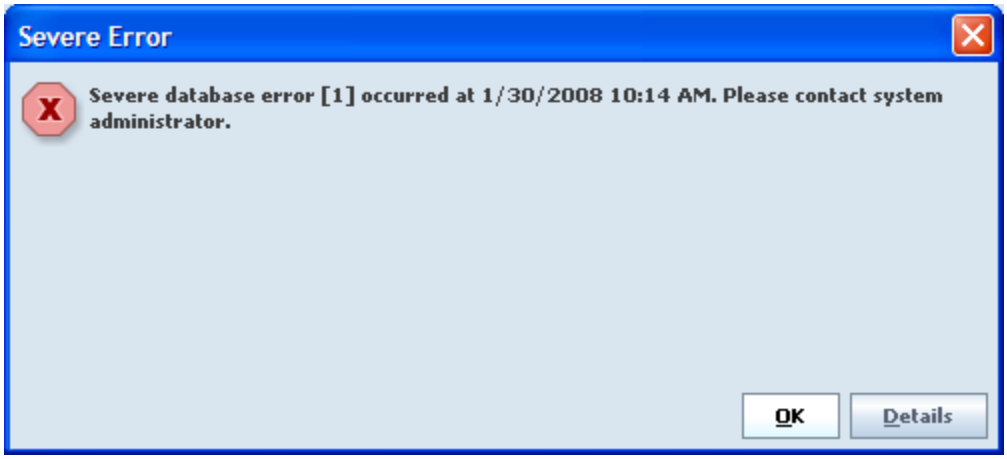
```
            itemTicket = ((ItemTicketWrapper) iterator.next()).getWrappedObject();
             try {
                  model.updateStockOnHand(itemTicket);
              } catch (BusinessException exception) {
                  displayException(exception);
              }
        }
        itemTicketTable.refreshTable();
}
```

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.



SimServerExceptions are thrown from the server layer and regardless of the details of the exception, only one message displays (see example). It includes an error number and the date time that the error occurred. This can be used to find the error within the server exception log. Pressing the details button displays the contents of the main exception.



Internationalization

No attempt should be made to internationalize exceptions at any layer. There should be no bundle references, no formatting, and so on. Error messages that are thrown across the EJB back to the client side, or are created on the client, will be internationalized and formatted by the exception handling framework on the client.

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.

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.

> **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: DirectDeliveryValidateLineItemRule

```
public RulesInfo execute(Shipment shipment, ShipmentLineItem lineItem) {
    if (shipment instanceof DirectDelivery ||
            shipment instanceof DirectDeliveryAsn) {
        try {
            if (!ItemHelper.isItemSuppliedBySupplier(
                    lineItem.getStockItem().getId(),
                    shipment.getFromLocation().getId())) {
                return RULE_FAILED;
             }
        } catch (Exception e) {
            LogService.error(this, "Failed executing rule", e);
        }
    }
    return RULE_PASSED;
}
```

Debugging

Please note that 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.

Example: Playing sound on wireless device...

```
try {
     AppGlobal.playSound("error");
} catch (Exception e) {
    LogService.warn(WirelessExceptionManager.class, "handleException()",
         "Unable to play sound ", null, e);
}
```

Example: RuleEngine

```
public void executeRule(String ruleGroupName, String ruleName, Object obj,
Object[] args) throws BusinessException {
```

```
  // we suspend because if we got here from a RIB injector, ClientContext is
already set
  ClientContext suspendedClientContext = ClientContext.suspendThreadInstance();
  // we set a new instance because if we got here from a SIM service no
ClientContext is set
  ClientContext.setThreadInstance(new ClientContext());
      try {
       ruleEngine.executeRule(ruleGroupName, ruleName, obj, args);
      } catch (RuleException ruleException) {
        BusinessException exception = buildBusinessException(ruleException);
        LogService.debug(this, ruleName + " failed for " + obj + ": " +
                  exception.getLocalizedMessage());
        throw exception;
    } finally {
       // restore, either to the suspended context, or null if none was set
       ClientContext.restoreThreadInstance(suspendedClientContext);
    }
}
```

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, etc). 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*>\bin\log\sim.log.

Logging is configured with files/prod/config/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 oc4j*>\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
```

Internationalization

Overview

Internationalization is the process of creating software that can be translated more 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.

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 PA_PRTY table (columns ED_CO, ED_LA). These columns are used to determine the country and language of the employee that logs in. Alternatively, the locale information for an employee may be filled in from an LDAP connection. Either way, 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: ItemMustBeRangedRule

```
public final class ItemMustBeRangedRule extends SimRule {

  private static final RulesInfo RULE_FAILED = new RulesInfo("Item is not
ranged.");
```

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.

Ex: ItemLookupPanel

```
  private RTextFieldEditor itemEditor = new RTextFieldEditor("Item");
  private RTextFieldEditor itemDescriptionEditor = new RTextFieldEditor("Item
Description");
```

Error Messages and Exception

Error messages are long explanations of some validation or event that took place 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 around the message you wish to display.

Dynamic Value Messages in Exceptions

BusinessExceptions are capable of handling dynamic values internally. The constructor that takes params 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 example below shows how

those numbers are placed into a parameters array and passed in the construction of a business exception.

Example: SequencingKeys

```
public class SequencingKeys {

    public static final String MAX_LOCATION_ERROR = "Too many locations within the
sequence. The maximum number of locations allowed is {0} and number of location in
the current sequence is {1}";
```

Example: LocationSequencer

```
public void resquenceAllLocations(List values) 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] = new Integer(values.size());
     throw new BusinessException(SequencingKeys.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: InventoryAdjustmentFilterPanel

```
private RDateFieldEditor fromDateEditor = new RDateFieldEditor("From Date");
private RDateFieldEditor toDateEditor = new RDateFieldEditor("To Date");

public void start() {
  try {
    model.loadFilter();
  } catch (BusinessException e) {
    displayException(e);
  }

  try {
    // Some Code
    InventoryAdjustmentQueryFilter filter = model.getFilter();

    fromDateEditor.setDate(filter.getFromDate());
    toDateEditor.setDate(filter.getToDate());
    // Additional Code
  } catch (Throwable exception) {
    displayException(exception);
  }
  assignFocusInScreen(fromDateEditor);
}

public void handleSearch() throws Exception {
  InventoryAdjustmentQueryFilter filter = model.getFilter();

  filter.setDateRange(fromDateEditor.getDate(), toDateEditor.getDate());
  filter.setInventoryAdjustmentId(adjustmentEditor.getTextOrNull());
  filter.setStatus((InventoryAdjustmentStatus) statusEditor.getSelectedItem());
  // 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.

Example: Date.cfg

```
# ENGLISH - AUSTRALIAN
#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.wirelessDate=ddMMyy
enAU.wirelessDisplay=DDMMYY

# ENGLISH - UNITED STATES
enUS.shortDate=M/d/yyyy
#enUS.mediumDate=MMM d, yyyy
#enUS.longDate=MMMM d, yyyy
#enUS.fullDate=EEEE, MMMM d, yyyy
enUS.monthPattern=MM-dd
enUS.wirelessDate=MMddyy
enUS.wirelessDisplay=MMDDYY

# JAPANESE - JAPAN
#jaJP.shortDate=yy/MM/dd
#jaJP.mediumDate=yyyy/MM/dd
#jaJP.longDate=yyyy/MM/dd
jaJP.monthPattern=MM-dd
jaJP.wirelessDate=yyMMdd
jaJP.wirelessDisplay=YYMMDD
```

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 basically of a BigDecimal amount and a String currencyCode (though Currency can only be set in the constructor). This is because once a money object exists, 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 SMoneyFieldEditor is a subclass that edits the SimMoney field directly. The following example demonstrates using the SMoneyFieldEditor.

Note that the 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.

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 forms, the display of labels is marked with a $[ ] indicator. In the Form<name>.java, the text within the brackets is wrapped by an AppGlobal.getString() call which translates the text.

Example: 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: 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 eventhandlers.

Key methods include:

Method	
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: 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: EventHandler_ContainerLookupDetail

```
protected void onFormOpen() {
  try {
    Container container = (Container)
UserContext.getValue(ContainerWirelessKeys.USER_CONTEXT_CONTAINER);
    // Some Code
    setFormData("asnNumber", container.getAsnID());
    setFormDate("eta", container.getETA());
    setFormDate("receiptDate", container.getReceiveDate());
    setFormData("receiptTime",
         LocaleWirelessUtility.formatTime(container.getReceiveDate()));
     setFormDecimal("totalCases", container.getNumberOfCases());
  } 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. Read the javadoc on WirelessUtility for complete descriptions.

Key methods include:

Method	
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
HandleException()	Handles displaying an exception (translating the message)
Alert()	Handles displaying an alert message (translating the message)

Here is an example of standard utility code using these internationalization methods.

Example: StockCountWirelessUtility

```
public static String getCountDescription(StockCountVO stockCountVO) {
  try {
    if (stockCountVO.getProductGroupType() ==
                  ProductGroupType.STOCK_COUNT_PROBLEM_LINE) {
      ProductGroup productGroup =
      ProductGroupHelper.readProductGroup(stockCountVO.getProductGroupId());
      StringBuffer buffer = new StringBuffer();
      buffer.append(getText("Problem Line ABBV"));
      buffer.append(getText(CommonWirelessKeys.LABEL_DELIMETER));
      buffer.append(productGroup.getDescription());
      return buffer.toString();
    }
    return stockCountVO.getDescription();
  } catch (Exception e) {
    return getText("*NO-DESC*");
  }
}
```

In the above example, business logic is required to create a stock count description. The utility uses the getText() helper method 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.

Available Languages

SIM supports the following languages:

- English
- French
- German
- Spanish
- Portuguese (Brazilian)
- Korean
- Chinese (Simplified)
- Chinese (Traditional)
- Russian
- Italian

Customizing Internationalization

The PC client provides administrative screens for adding and altering translations.

PC/User Interface Development

Overview

The SIM PC client is a Swing application. It is launched using WebStart and a browser. It communicates with the SIM server through Enterprise Java Beans (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.

Coding Guidelines

- Clients should always access services through the <*name*>Helper class.
- Do not modify framework related code.

PC Client Architecture

The client architecture is broken into five layers, only three of which are worked on during application development.

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 singular responsibility is to interact with the navigation system and delegate its actions to the panel [ex. ItemLookupScreen].

<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 (for example, ItemLookupPanel).

<name>Model

The model is where all access to the service layer takes place as well as where any complex logic that needs to take place on the client is written.

EJB Service

An EJB service is accessed through a service Helper class and should only be done from within a model class. There are very few exceptions to this design pattern, though they do exist. (for example, ItemHelper).

Navigation

There is both configurable navigation and hard-coded navigation within the SIM application.

External Configurable Navigation

The primary means of navigation in the application is through pressing the menu buttons displayed at the top of the screen. These menu buttons are determined by the navigation.xml file located at files/prod/config/simgui/navigation.xml. Adding, editing, and removing buttons for customization must be done within this navigation file.

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 is an excellent example of this. Because it can be reached through screens we do not want to return to via the standard framework, very specific navigation is coded for the screen.

For example, in the validDelivery() method, only if the direct delivery is successfully cancelled does the application return to the previous screen. This is done using the SimNavigation.BACK value as the command to navigate to. SimNavigation contains a series of static variables that represent all the menu options in the system (plus BACK).

```
private boolean validDelivery() throws Exception {
  if (panel.isDeliveryEmpty()) {
    if (RConfirmUtility.confirm("Item Delete Confirmation",
                DirectDeliveryConstants.NO_ROWS_REMAINING)) {
       panel.cancelEmptyDelivery();
       navigate(SimNavigation.BACK);
       return false;
     }
  }
  return true;
```

```
}
```

Another very common usage of navigate is to jump to a lookup screen. This is also done in DirectDeliveryDetailScreen in the buildItemSearchListener() method. In this case SimScreenName.ITEM_LOOKUP_SCREEN is used as the navigation command. The class SimScreenName contains static variables for all the screens that can be navigated to within the code. The navigate() method simply takes the full classpath name of the <name>Screen to navigate to.

```
private SearchListener buildItemSearchListener() {
    return new SearchListener() {
        public void search(SearchReceiver receiver) {
            searchReceiver = receiver;
           // Setup Repository Information
            navigate(SimScreenName.ITEM_LOOKUP_SCREEN);
        }
    };
}
```

Screens

The following is a break-down of the structure of screens. Use these coding guidelines when creating or modifying an existing screen. The only responsibility of screens is to interact with the navigation framework.

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.
2. Implement the getScreenName() method. This should return the title of the screen to be displayed. This is a simple text label. See internationalization documentation for how the translation of this text is handled.

   ```
   public class ItemLookupScreen extends SimScreen {

       private ItemLookupPanel panel = new ItemLookupPanel();

       public ItemLookupScreen() {
           add(panel);
       }

       public String getScreenName() {
           return "Item Lookup";
       }
   }
   ```
3. Implement the init() method. This method is only called the very first time the screen is instantiated. There will seldom need to be code placed in this method. Initialization code can never throw an exception.
4. 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 init() { }
   public void stop() { }

   // If user is back from Item Detail, and clicked on Use Item, redirect the
   user to the origin.
   public void start() {
     if (RepositoryManager.getStateObject(ItemConstants.SELECTED_ITEM) != null) {
       try {
   ```

```
        panel.handleDone();
        navigateLater(SimNavigation.BACK);
        return;
      } catch (Throwable exception) {
        displayException(exception);
      }
    }
    showMenu(SimNavigation.SEARCH);
    panel.start();
  }
```

5. Implement the start() method. This method is called whenever the screen is navigated to. As shown here, sometimes the state of the application determines what should take place.

   Of interest are the last two lines of code, which should be found in 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.

6. 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).

7. There are two new methods isStartable() and isStoppable() that were added to the framework for SIM 13.0. They are checked before start() and stop() are called respectively. 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() return false, the screen will not allow itself to be navigated away from.

8. 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 pressed it. The implementation of this method should determine which button is pressed and delegate to the appropriate panel method.

```
public void navigationEvent(NavigationEvent event) {
  String command = event.getCommand();
  try {
    if (command.equals(SimNavigation.DONE)) {
        panel.handleDone();
    } else if (command.equals(SimNavigation.SEARCH)) {
        panel.handleSearch();
    } else if (command.equals(SimNavigation.RESET)) {
        panel.handleReset();
    }
  } catch (Throwable exception) {
        displayException(panel, event, exception);
  }
}
```

Handling Navigation Events

After the NavigationEvent from a menu button is processed by the screen, the application level framework uses the content of the event to determine if navigation takes place. Sometimes, during the processing of an event, the state of the situation changes so that the navigation should not take place. To stop the framework from using the event, simple call event.consume(). A consumed event will be ignored by the navigation engine.

Hiding a Menu Button Programmatically

The state of the application as a screen opens sometimes determines that certain menu options do not appear on the screen. In this case, simply call removeNavButton() with the identifier of the button to be removed. The only caveat to remember is that showMenu() must be executed prior to this or the button will not be available to remove.

Example: InventoryAdjustmentDetailScreen

```
public void start() {
  // Some repository code
  showMenu();

  if (panel.isAdjustmentPending()) {
     panel.setTableEditable(true);
     removeNavButton(SimNavigation.ADD_ITEM);
     removeNavButton(SimNavigation.DELETE);
  } else if (panel.isAdjustmentCompleted()) {
     panel.setTableEditable(false);
     removeNavButton(SimNavigation.ADD_ITEM);
     removeNavButton(SimNavigation.DELETE);
     removeNavButton(SimNavigation.CANCEL);
  }
}
```

Panels

The following is a break-down of the structure of panels. Use these coding guidelines when creating or modifying an existing panel.

1. Declare the panel. screens, panels, and models should be declared in the package oracle.retail.sim.shared.swing.<name> where <name> is the functional area the screen belongs to.
2. The User Interface panel must extend the screenpanel.

   ```
   public class ItemLookupPanel extends ScreenPanel implements REventListener {

       private ItemLookupModel model = new ItemLookupModel();
   ```

3. 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. For a list of common editors, check the Editor section in this document. Each editor is usually constructed with its label. The label should be entered as the text to be displayed. See the internationalization documentation for how the label is used.

   SimTable is the most common used type of table in the application. SimTable should be placed in a SimTablePane before being placed on the screen. See the SimTable javadoc or the SimTable section of this document for further information.

   ```
   private RTextFieldEditor itemEditor = new RTextFieldEditor("Item");
   private RTextFieldEditor itemDescriptionEditor
     = new RTextFieldEditor("Item Description");
   private RTextFieldEditor supplierEditor = new RTextFieldEditor("Supplier");
   private RTextFieldEditor supplierNameEditor
     = new RTextFieldEditor("Supplier Name");
   private ItemHierarchyPanel itemHierarchyPanel = new ItemHierarchyPanel();
   private RIntegerFieldEditor limitEditor
     = new RIntegerFieldEditor("Search Limit");
   private RCheckBoxEditor nonRangedEditor
     = new RCheckBoxEditor("Include Non-Ranged");

   private SimTable itemTable = new SimTable(new ItemTableDefinition());
   private SimTablePane itemPane = new SimTablePane(itemTable);
   ```

4. Declare Constructor. The constructor of a screen never takes a parameter. In order to keep a consistent pattern, the constructor should call an initialize method and layout method to setup all the UI components and lay them out on the panel.

```
public ItemLookupPanel() {
    initializePanel();
    layoutPanel();
}
```

5. Initialize the Panel. These two methods are not required, but should be a pattern followed in each panel. The initializePanel() method should setup the property settings on editors and tables. In the below example, the identifier for all the editors is assigned as well as registering an action on the table. The layoutPanel() method is used to layout all the components on the panel. REditorPanels are useful tools for gathering editors in groups. GridBagLayout is the preferred layout for complex organization of sub-panels. The GridTool class allows layout constraints to be created using shorthand numeric values.

```
private void initializePanel() {
  itemEditor.setIdentifier(SimName.ITEM_ID);
  itemDescriptionEditor.setIdentifier(SimName.ITEM_DESCRIPTION);
  supplierEditor.setIdentifier(SimName.SUPPLIER_ID);
  supplierNameEditor.setIdentifier(SimName.SUPPLIER_NAME);
  limitEditor.setIdentifier(SimName.ITEM_SEARCH_LIMIT);
  limitEditor.setSizeType(EditorConstants.SMALL);

  itemTable.setTableEditable(false);
  itemTable.setSingleRowSelectionMode();
  itemTable.registerDoubleClickAction(this, ITEM_SELECTED);
}

private void layoutPanel() {
  REditorPanel leftPanel = new REditorPanel(5);
  leftPanel.add(itemEditor);
  leftPanel.add(itemDescriptionEditor);
  leftPanel.add(supplierEditor);
  leftPanel.add(supplierNameEditor);
  leftPanel.add(nonRangedEditor);

  REditorPanel limitPanel = new REditorPanel(1, 2);
  limitPanel.add(limitEditor);
  limitPanel.add(nonRangedEditor);

  LayoutUtility.alignPanels(itemHierarchyPanel, limitPanel);

  RPanel mainPanel = new RPanel(new GridBagLayout());
  mainPanel.add(leftPanel, GridTool.constraints(0, 0, 1, 2, 1, 0, 0, 3, 0, 0, 0, 0));
  mainPanel.add(itemHierarchyPanel, GridTool.constraints(1, 0, 1, 1, 0, 0, 0, 3, 0, 0, 0,
0));
  mainPanel.add(limitPanel, GridTool.constraints(1, 1, 1, 1, 0, 0, 0, 3, 0, 0, 0, 0));
  mainPanel.add(itemPane, GridTool.constraints(0, 2, 2, 1, 1, 1, 0, 3, 5, 0, 0, 0));

  setContentPane(mainPanel);
}
```

6. Start the Panel. A start() method should be created so that the screen may 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. The component passed to this method will receive focus when the panel appears.

```
public void start() {
  if (RepositoryManager.getStateObject(ItemConstants.ITEM_FILTER_IN_PROGRESS) == null) {
    clearScreen();
  }
```

```
    if (RepositoryManager.getStateObject(ItemConstants.SELECTED_SUPPLIER) != null) {
      Supplier supplier = (Supplier)
         RepositoryManager.getStateObject(ItemConstants.SELECTED_SUPPLIER);
      RepositoryManager.removeStateObject(ItemConstants.SELECTED_SUPPLIER);
       supplierEditor.setText(supplier.getId());
    }

    try {
      itemHierarchyPanel.loadDepartments();
    } catch (Exception exception) {
      displayException(exception);
    }
    assignFocusInScreen(itemEditor);
  }
```

Models

The following is a break-down of the structure of models. Use these coding guidelines when creating or modifying an existing screen model.

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.

   Example: InventoryAdjustmentDetailModel

   ```
   public class InventoryAdjustmentDetailModel extends SimScreenModel {

      private InventoryAdjustment inventoryAdjustment = null;
   ```

2. Add all the functional methods required by the panel to communicate with the server or manipulate data on a logic level. These model methods represent access to the information represented by the model. Here is an example of common methods found in a model.

   Example: InventoryAdjustmentDetailModel

   ```
   public void loadAdjustment() {
     inventoryAdjustment = (InventoryAdjustment) RepositoryManager
          .getStateObject(InventoryConstants.SELECTED_INVENTORY_ADJUSTMENT);
   }

   public void createAdjustment() throws BusinessException {
     inventoryAdjustment = BOFactory.createInventoryAdjustment(getStoreId());
     inventoryAdjustment.setEmployeeId(getEmployeeId());
   }

   public boolean isAdjustmentPending() {
   return
   InventoryAdjustmentStatus.PENDING.equals(inventoryAdjustment.getStatus());
   }

   public void updateNewAdjustment() throws Exception {
     InventoryAdjustmentHelper.createInventoryAdjustment(inventoryAdjustment,
                          InventoryAdjustmentSource.MANUAL);
   }
   ```

Features of the Model

The SimScreenModel superclass contains several features that can be used by all subclasses. The employee/employee id and store/store id currently logged on can be retrieved. System configuration and store configuration values can be retrieved. Most importantly, access to the activity locking functionality is available only within the model. Methods available are releaseLock(), checkLock() and getLock(). Here are several example methods of accessing locks through a model.

Example: InventoryAdjustmentDetailModel

```
public boolean obtainAdjustmentLock() throws Exception {
  if (isAdjustmentPending()) {
    return getLock(inventoryAdjustment.getId(), ADJUSTMENT_LOCK_TYPE, true);
  }
  return true;
}

public void releaseAdjustmentLock() throws Exception {
  if (isAdjustmentPending()) {
    releaseLock();
  }
}

public boolean checkAdjustmentLock() throws Exception {
  if (isAdjustmentPending()) {
    return checkLock(inventoryAdjustment.getId());
  }
  return true;
}
```

Dialog Windows

There are three basic type of dialogs displayed in SIM: messages, errors and confirmation.

Errors/Exceptions

The developer never instantiates or creates an error dialog. This type of dialog is 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 exceptions.

Example: ItemLookupPanel

```
try {
  itemHierarchyPanel.loadDepartments();
} catch (Exception exception) {
  displayException(exception);
}
```

Example: ItemLookupScreen

```
public void navigationEvent(NavigationEvent event) {
  String command = event.getCommand();
  try {
    if (command.equals(SimNavigation.DONE)) {
      panel.handleDone();
    } else if (command.equals(SimNavigation.SEARCH)) {
      panel.handleSearch();
    } else if (command.equals(SimNavigation.RESET)) {
      panel.handleReset();
    }
  } catch (Throwable exception) {
```

```
    displayException(panel, event, exception);
  }
}
```

Example: InventoryMenuScreen

```
private void doStoreOrder(NavigationEvent event) {
  try {
     StoreOrderHelper.pingExternalService("test");
  } catch (Throwable exception) {
     displayException(LoginConstants.STORE_ORDER_BAD_CONNECT);
RepositoryManager.removeStateObject(StoreOrderConstants.STORE_ORDER_QUERY_FILTER);
    event.consume();
  }
}
```

Messages

The developer never instantiates message windows, but uses the methods available at the panel level to display messages. 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.

Example: ItemTicketListPanel

```
private void printTickets() throws Exception {
  try {
     List wrappers = itemTicketTable.getAllSelectedRowData();
     int failedCount = model.printTickets(wrappers);
     Object[] params = new Object[2];
     params[0] = new Integer(wrappers.size() - failedCount);
     params[1] = new Integer(failedCount);

    displayMessage(Translator.getMessage(
              ItemTicketConstants.TICKETS_PRINTED_LONG, params));
} catch (BusinessException exception) {
    displayException(exception);
} finally {
    itemTicketTable.setRows(model.findItemTickets());
}
}
```

Example: ContainerLookupPanel

```
public void handleSearch() throws Exception {
  // Some Code…
  if (containerTable.isEmpty()) {
     displayWarning(ContainerConstants.NO_RECORDS);
     assignFocusInScreen(containerIdEditor);
     return;
  }
  // Additional Code…
}
```

Confirmation

Confirmation Dialogs are Ok/Cancel or Yes/No dialogs that allow the user to make decisions. Indeed, confirmation dialogs are not instantiated either. A utility class (RConfirmUtility) is supplied to make the display of confirmation dialogs easier. Please read the javadoc on RConfirmUtility for a complete description of the options available.

The confirm() method returns true if the option was confirmed, false if it was not.

Example: ReturnDetailPanel

```
public boolean cancelReturn() throws Exception {
  if (RConfirmUtility.confirm("Delete Return Confirmation",
```

```
                ReturnsConstants.CANCEL_RETURN_CONFIRM)) {
    model.cancelReturn();
    return true;
  }
  return false;
}
```

Example: MacroSequenceListPanel – The confirmation button labels are passed as parameters.

```
public void handleApplyClassList() throws Exception {
  // Some Code…

  LocationArea area = LocationArea.BACKROOM;
  if (RConfirmUtility.confirm("Select Area",
     SequencingConstants.SHOPFLOOR_OR_BACKGROOM, "Shopfloor", "Backroom")) {
    area = LocationArea.SHOPFLOOR;
  }

  try {
     model.applyClassList(area);
  } catch (BusinessException exception) {
     displayException(exception);
  }
  locationTable.setRows(model.findLocations());
}
```

Editors

The following is a brief list of common editors and what they do. Editors are the only kinds of components that should be displayed in a SIM screen. If a new editor is needed, the technical architect should be informed.

Editor	Description
RCheckBoxEditor	A check box that may 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 window 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.
RQuantityEditor	A text field that only edits Quantity values.
RPasswordFieldEditor	A text field that allows the hidden entry of a password.
RPercentFieldEditor	A text field that only allows percent values to be entered.
RRadioButtonEditor	An editor that allows a set of radio 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.

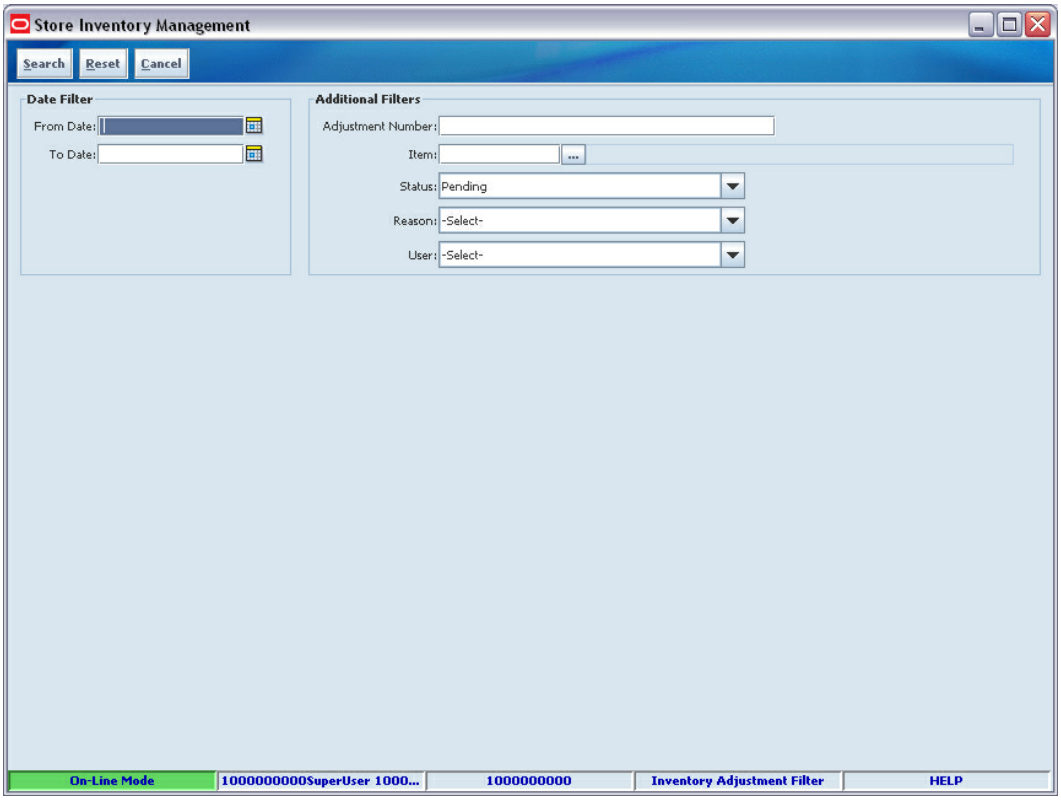
Editor	Description
RSearchFieldEditor	A text field that allows the user to enter a value or search for one.
RTextAreaEditor	A large text area where large quantities of text may be edited.
RTextFieldEditor	A single line text field where small quantities of text may be edited.

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 InventoryAdjustmentFilterPanel has a search editor for its item value that will be used as our example.

If an ID is entered, the editor automatically searches for the whole object. When the button is clicked, it navigates to a screen where an object is chosen. Upon return, the data of the object is displayed.

Example: InventoryAdjustmentFilterPanel



Using a Search Editor

Use the following procedure to use a search editor.

1. Declare the search editor.

```
private RSearchFieldEditor editor = new RSearchFieldEditor("Item");
```

2. Set all the properties of the search editor. Five basic properties are required for a search editor to function: identifier, entry displayer, value displayer, search processor, and search listener.
   - The identifier is a component's id and can be used to reference the component uniquely.
   - The entry displayer determines how the ID half of the search editor should display its data. For more information on Displayers, see "Displayers".
   - The value displayer determines how the description half of the search editor should display its data.
   - 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.
   - The search listener is a class that implements the SearchListener interface. This class will be called when the search button is pressed.

```
editor.setIdentifier(SimName.ITEM_ID);
editor.setEntryDisplayer(new AttributeDisplayer("rin"));
editor.setValueDisplayer(new StockItemDisplayer());
editor.setSearchProcessor(new ItemSearchProcessor(allowNonRanged));
editor.setSearchListener(buildItemSearchListener());
```

3. Building a search listener. A search listener needs to be declared independently for each search editor that is used. The search editor passes itself into the search() method as a SearchReceiver. The Panel needs to store this value so when start() is called after the screen returns from the lookup, the correct search receiver can have its data assigned. This search() method is triggered when the search button is pressed in the editor.

```
private SearchListener buildItemSearchListener() {
  return new SearchListener() {
    public void search(SearchReceiver receiver) {
       searchReceiver = receiver;
       RepositoryManager.addStateObject(
           InventoryConstants.ITEM_LOOKUP_IN_PROGRESS, Boolean.TRUE);
       RepositoryManager.addStateObject(ItemConstants.ITEM_LOOKUP_TYPE,
           ItemConstants.VALUE_ITEM_LOOKUP);
        navigate(SimScreenName.ITEM_LOOKUP_SCREEN);
    }
  };
```

4. The following is an example of how after retrieving the data stored in the global repository from the search, `setData()` is called on the SearchReceiver.

```
public void startFromItemLookup() {

RepositoryManager.removeStateObject(InventoryConstants.ITEM_LOOKUP_IN_PROGRESS
);
  ItemVO item = (ItemVO)
          RepositoryManager.getStateObject(ItemConstants.SELECTED_ITEM);
  RepositoryManager.removeStateObject(ItemConstants.SELECTED_ITEM);
  if (item != null) {
  searchReceiver.setData(item);
  }
  assignFocusInScreen(itemEditor);
}
```

SimTable

SimTable is an extension of JTable that allows the display and editing of cells and data. It contains a great deal of advanced options not available with the standard table. It is also specifically designed to handle common design patterns in SIM.

Using a SimTable

1. Declare the SimTable globally within the panel. It requires passing in a table definition (see SimTableDefinition for details). The SimTable should be placed within a SimTablePane and the pane added to the layout. SimTablePane supplies the scrollbars for the table.

   Example: ItemLookupPanel

   ```
   private SimTable itemTable = new SimTable(new ItemTableDefinition());
   private SimTablePane itemPane = new SimTablePane(itemTable);
   ```

2. In the initializePanel() method, set desired properties on the table. In the case of ItemLookupPanel, the table editable property is set to false and an action is registered for when a row is double-clicked.

   ```
   itemTable.setTableEditable(false);
   itemTable.setSingleRowSelectionMode();
   itemTable.registerDoubleClickAction(this, ITEM_SELECTED);
   ```

3. Assigning data to the table is simply a matter of passing a collection of objects to be displayed. The setRows() method will automatically replace all the content of the table. The objects must all be of the class type declared in the table definition (see SimTableDefinition).

   ```
   itemTable.setRows(model.findItemVOs(searchFilter));
   ```

4. Retrieving information from the table is as simple as using the get***Data methods available. Methods ending in Data retrieve the object represented by the row, so getSelectedRow() returns the row index whereas getSelectedRowData() returns the row object.

   ```
   ItemVO item = (ItemVO) itemTable.getSelectedRowData();
   ```

SimTableDefinition

A table definition is the core of how a table works. To create a table definition, implement SimTableDefinition. The getDataClass() method must return the class of objects that the table will display in each row. The attributes define the list of columns that the table will display. The getAttributes() method returns an array of SimTableAttribute objects that defines each column. The getSortAttributes() method defines the default sort column of the table.

Example: ItemLookupPanel

```
private static class ItemTableDefinition extends SimTableDefinition {

  public Class getDataClass() {
    return ItemVO.class;
  }

  public List getSortAttributes() {
    if (SimConfig.isItemShortDescription()) {
      return Collections.singletonList(
                  new SimTableSortAttribute("shortDescription"));
    }
    return Collections.singletonList(
                  new SimTableSortAttribute("longDescription"));
  }

public List getAttributes() {
  List attributes = new ArrayList();
  attributes.add(new SimTableAttribute("Item", "id"));
  if (SimConfig.isItemShortDescription()) {
   attributes.add(new SimTableAttribute("Item Description", "shortDescription"));
  } else {
   attributes.add(new SimTableAttribute("Item Description", "longDescription"));
  }
  attributes.add(new SimTableAttribute("Supplier", "supplierVO.id"));
  attributes.add(new SimTableAttribute("Supplier Name", "supplierVO.name"));
  attributes.add(new SimTableAttribute("Dept.", "departmentName"));
  attributes.add(new SimTableAttribute("Class", "className"));
  attributes.add(new SimTableAttribute("Sub-Class", "subclassName"));
  return attributes;
  }
```

In some cases, a single row of the table actually represents data that is original 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. See “Wrappers”.

SimTableAttribute

A SimTableAttribute represents exactly one column within the SimTableDefinition. There are many different properties that can be assigned to the object. The basics are outlined below.

Example: InventoryAdjustmentDetailPanel

```
public List getAttributes() {
  List attributes = new ArrayList();
  attributes.add(new SimTableAttribute("Item", "stockItem",
      new AttributeDisplayer("id"), stockItemEditor));
  attributes.add(new SimTableAttribute("Item Description", "description"));
  attributes.add(new SimTableAttribute("UOM", "unitOfMeasure",
      new UOMDisplayer(), new UOMTableEditor()));
  attributes.add(new SimTableAttribute("Pack Size", "packSize"));
  attributes.add(new SimTableAttribute("Quantity", "qtyBasedOnUOM",
      new QuantityDisplayer(), new LineItemQuantityTableEditor()));
  attributes.add(new SimTableAttribute("Reason", "reason",
      new AttributeDisplayer("description"), reasonTableEditor));
  return attributes;
```

1. 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.
2. Attribute – This 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.
3. 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 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 (see “Displayers”). In the example below, an AttributeDisplayer is assigned to the “stockItem” attribute and a UOMDisplayer is assigned to the “unitOfMeasure” attribute.
4. 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 GeneralQuantityTableEditor is assigned, if 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 (see “TableEditors”). In the example below, a StockItemTableEditor is assigned to the “stockItem” attribute and a UOMTableEditor is assigned to the “unitOfMeasure” attribute. The StockItemTableEditor is declared at the class level so it can modified by class code.
5. 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 contains useful generic displayers.

Creating a New Displayer

Use the following procedure to create a new displayer is needed:

1. Check all previously existing displayers. Many of the generic displayers can handle formatting different objects (especially note AttributeDisplayer, DualAttributeDisplayer and DefaultDisplayer).
2. Design the new displayer and extend the appropriate superclass (often AbstractDisplayer).
3. Implement the getDisplayText() methods.

Example: 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();
        }
        if (object instanceof FullStore) {
            FullStore store = (FullStore) object;
            return store.getId() + separator + store.getName();
        }
        return object.toString();
    }
```

The Displayable Interface

There is a new interface in SIM 13.0 named Displayable with a single method toDisplayString(). This is intended to be implemented by wrappers, business objects and any other type of object that may need to have a "display" value, but which the toString() method is primarily reserved for debugging. At the moment, there are no objects that implement Displayable. However, for convenience, here is Supplier if Displayable was implemented. The new framework in the closed packages is already set up to recognize and use Displayable. For example, if you drop a bunch of Displayable objects into an RComboBoxEditor, it automatically calls toDisplayString() to determine what to display.

```
public abstract class Supplier extends Source implements Displayable {
   //All The Class Code

    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("Supplier: Rin(");
        buffer.append(getRin());
        buffer.append(") Name(");
        buffer.append(getName());
        buffer.append(") Status(");
        buffer.append(status);
        return buffer.toString();
    }

    public String toDisplayString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append(getRin());
        buffer.append(" - ");
        buffer.append(getName());
        return buffer.toString();
    }
```

TableEditors

Like the editors designed to be used on screens, table editors are advanced widgets that are designed to be used within table cells. It all begins with the SimTableEditor interface, which the SimTable uses to control editing within its cells. There are many generic table editors located within the oracle.retail.sim.closed.swing.tableeditor package. If these do not meet your requirements, there are SIM specific table editors within the package oracle.retail.sim.shared.swing.tableeditor. If these do not meet your needs, you will have to write your own table editor.

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 should be done by a senior developer for any new table editors.

SimTableErrorTask

There is a new API on SimTable that is very useful: setColumnEditorErrorTask(). It takes two parameters: an attribute and an error task. Whenever editing the attribute triggers an error within the table, the error task is executed. This allows logic to take place within the SimTable's framework for handling exceptions.

Example: DirectDeliveryDetailPanel

```
itemTable.getTableEditor(Integer.class).addTableEditorListener(buildIntegerListene
r());
itemTable.setColumnEditorErrorTask("stockItem", new StockItemErrorTask());
itemTable.setTableEditable(true);
```

In the provided example of using an error task, the StockItemErrorTask is being assigned to the attribute "stockItem". On the direct delivery detail panel, when the stock item is altered in the table, the table attempts to update the underlying data object. If an exception takes place, the source exception will be delivery to the handleException() method of the SimTableErrorTask.

Note how the code can take a different route depending on the type of error. In the case below, it checks if the original cause was a BusinessException. It then checks if the BusinessException occurred because a duplicate line on the shipment was created by the stock item. It then hunts through the original table and selects the original row that the stock item exists in.

If it is not that specific type of exception, it simply displays the exception and continues on.

Example: DirectDeliveryDetailPanel

```
private class StockItemErrorTask extends SimTableErrorTask {

  protected void handleException(SimTableException exception) {
    Throwable originalException = exception.getCause();

    if (originalException instanceof BusinessException) {
       BusinessException ruleException
           = (BusinessException) originalException;
       UIException uiException
           = new UIException(ruleException.getLocalizedMessage());

      itemTable.stopEditing();

      if (ruleException.containsMessage(
            ShipmentLineItemDupItemNotAllowedRule.ERROR_MESSAGE)) {
        uiException
         = new UIException(DirectDeliveryConstants.ITEM_ALREADY_EXISTS);
      }

       UIStatusUtility.displayException(this, uiException);

       StockItem stockItem = (Stockable) exception.getValue();
       DirectDeliveryItemWrapper wrapper = null;
       int row = 0;
       for (Iterator iterator = itemTable.getAllRowData().iterator();
                              iterator.hasNext();) {
         wrapper = (DirectDeliveryItemWrapper) iterator.next();

          if (stockItem.equals(wrapper.getStockItem())) {
            itemTable.setRowSelectionInterval(row, row);
       itemTable.scrollRectToVisible(itemTable.getCellRect(row, 1, false));
            break;
          }
```

```
            row++;
          }
          return;
        }
        UIStatusUtility.displayException(this, exception);
      }
```

Table Wrappers and Value Objects

When creating a table definition, the getDataClass() is used to specify what object is being displayed. Quite 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 it is gathered and displayed on the screen.

There are two methods of dealing with a row that does not cleanly map to a business objects: wrappers and value objects.

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 UI client code can live that does not belong to the behavior of the business object.

Bad Example: DirectDeliveryDetailPanel

```
private static class DirectDeliveryItemDefinition implements SimTableDefinition {

  public Class getDataClass() {
    return ReceiptLineItem.class;
  }
```

Example: DirectDeliveryDetailPanel

```
public class DirectDeliveryItemDefinition implements SimTableDefinition {

    public Class getDataClass() {
        return DirectDeliveryItemWrapper.class;
    }
```

Creating a Wrapper

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 object(s). Note how the wrapper contains both a shipment and receipt line item. This makes it convenient to create one table row API that accesses both objects.

Example: DirectDeliveryItemWrapper

```
public class DirectDeliveryItemWrapper implements LineItemWrapper {

    private Shipment shipment = null;
    private ReceiptLineItem lineItem = null;

    private boolean createdThisSession = true;

    public DirectDeliveryItemWrapper(Shipment shipment) {
        this.shipment = shipment;
    }

    public DirectDeliveryItemWrapper(Receipt receipt, ReceiptLineItem
            receiptLineItem) {
        this.shipment = receipt.getShipment();
        this.lineItem = receiptLineItem;
    }
```

In the above example, note the declaration of createdThisSession. This type of indicator is not really something to be placed on a business object, but makes perfect sense in the temporary wrapper created for the UI.

Example: DirectDeliveryItemWrapper

```
public boolean isShipmentSaved() {
  return (shipment.getStatus().equals(Shipment.RECEIVED) ||
      shipment.getStatus().equals(Shipment.IN_PROGRESS));
```

In the above example, this method is useful on the UI Wrapper API, but may not fit the type of information you want on the API of the business object for Shipment. If you bring up the source code for this wrapper, you should also look at the getUnitCost() method of the same wrapper. It is another good example of logic that should live on the client wrapper for convenience, but the business object should definitely not implement this logic.

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 at all as the value object is considered unchangeable. The ItemLookupPanel and ItemVO is a good example of this pattern.

Example: ItemLookupPanel

```
private static class ItemTableDefinition extends SimTableDefinition {

  public Class getDataClass() {
    return ItemVO.class;
```

Example: ItemVO

```
public class ItemVO implements Serializable {
    private static final long serialVersionUID = -8373897294315835558L;

    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;

    public ItemVO(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public String getShortDescription() {
        return shortDescription;
    }

    public String getLongDescription() {
        return longDescription;
    }
```

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 thing 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, you may want to take 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

There are several steps 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: DirectDeliveryCreatePanel

```
private static final String ITEM_MODIFIED = "Item.modified";
private static final String SUPPLIER_MODIFIED = "Supplier.modified";

private void initializePanel() {
    itemEditor.registerAction(this, ITEM_MODIFIED);
    itemSupplierEditor.registerAction(this, SUPPLIER_MODIFIED);
}
```

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. It helps with debugging as well.

   Example: DirectDeliveryCreatePanel

```
public void performActionEvent(RActionEvent event) {
    String command = event.getEventCommand();
    try {
        if (command.equals(ITEM_MODIFIED)) {
            doItemModified();
        } else if (command.equals(SUPPLIER_MODIFIED)) {
            doSupplierModified();
        }
    } catch (Throwable exception) {
        displayException(exception);
    }
}
```

3. Implement the action logic.

   Simply implement the private method that executes the logic associated with the editor.

   Example: DirectDeliveryCreatePanel

```
private void doSupplierModified() throws Exception {
    Supplier supplier = (Supplier) supplierEditor.getData();
    if (supplier != null) {
      supplierPOEditor.setItems(model.findSupplierPurchaseOrders(supplier));
    }
    validateEditorState();
}
```

Table Editor Actions

Editors within tables can be "listened" to in order to receive events, but they are handled in an entirely different manner.

First, 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. DirectDeliveryCreatePanel has an excellent example of both.

Example: DirectDeliveryCreatePanel

```
private StockableTableEditor stockableEditor = new StockableTableEditor();

private void initializePanel() {
    stockableEditor.addTableEditorListener(buildStockableListener());
    itemTable.getTableEditor(Integer.class).addTableEditorListener(buildIntegerLis
tener());
}
```

The key here is that a listener was "built" in both cases. Using a build<name>Listener method is a way to get the code separated into a convenient method rather than declaring inline where it might obfuscate what is taking place. Thus, the next step is to build the listener. In the example below, every time the stock item value changes within the table, the checkFields() method is called within the panel.

Example: DirectDeliveryCreatePanel

```
private SimTableEditorListener buildStockItemListener() {
    return new SimTableEditorListener() {
        public void performTableEditorEvent(SimTableEditorEvent event) {
            checkFields();
        }
    };
}
```

Key Classes

The following list is the key classes within the PC UI layer.

- BasicDisplayer
- Displayable
- Displayer
- RConfirmUtility
- REventListener
- Screen
- ScreenPanel
- SimNavigation
- SimScreen
- SimScreenName
- SimScreenModel
- SimTable
- SimTableAttribute
- SimTableEditor

Customizing the User Interface

It is suggested that if a screen needs to be customized for the PC, that the two top-level classes <name>Screen and <name>Panel be copied into the custom workspace while the panel is assigned a new custom model that extends the previous model.

Once the changes are made, these three classes should be placed in a custom JAR that is first on the classpath.

For example, if extending the ItemLookupScreen, then both ItemLookupScreen and ItemLookupPanel should be copied to the customization workspace area. The same package organization must be kept within this new work area.

A new class MyItemLookupModel should be created that extends the original ItemLookupModel. This new model should be the one used within the customized panel code.

Example: MyItemLookupModel

```
public class MyItemLookupModel extends ItemLookupModel {
```

Example: ItemLookupPanel (customized)

```
public class ItemLookupPanel extends ScreenPanel implements REventListener {
  private static final long serialVersionUID = -3031044901716067689L;

  private MyItemLookupModel model = new MyItemLookupModel ();
```

Once the classes are in a custom workspace, new editors, tables, buttons and functional logic can be added.

Attempting to put as much functional logic as possible in the new model assists in future releases. The superclass model can be updated on the “base” path without affecting the extended class at all. Unfortunately, any changes to panel and screens will need to be re-applied in all future releases.

Customizing Navigation

The navigation.xml file determines all the buttons that display in the top navigation bar as well as what occurs after the button is pressed and the panel notified.

To modify the navigation.xml file, it will have to be removed from the JAR it is in, modified it, and placed back in the JAR.

Here is a piece of the navigation.xml.

Example: navigation.xml.

```
<task>
  <task_name>oracle.retail.sim.shared.swing.item.ItemLookupScreen</task_name>
    <default_task_item>Default</default_task_item>
    <task_item>
        <task_item_name>Done</task_item_name>
        <task_item_permission>0</task_item_permission>
        <task_item_command>BACK</task_item_command>
        <task_item_duplicate>false</task_item_duplicate>
    </task_item>
    <task_item>
        <task_item_name>Use Item</task_item_name>
        <task_item_permission>0</task_item_permission>
        <task_item_command>BACK</task_item_command>
        <task_item_duplicate>false</task_item_duplicate>
    </task_item>
    <task_item>
        <task_item_name>Search</task_item_name>
        <task_item_permission>0</task_item_permission>
        <task_item_command>NONE</task_item_command>
        <task_item_duplicate>false</task_item_duplicate>
    </task_item>
    <task_item>
        <task_item_name>Reset</task_item_name>
        <task_item_permission>0</task_item_permission>
        <task_item_command>NONE</task_item_command>
        <task_item_duplicate>false</task_item_duplicate>
    </task_item>
    <task_item>
        <task_item_name>Cancel</task_item_name>
        <task_item_permission>0</task_item_permission>
        <task_item_command>BACK</task_item_command>
        <task_item_duplicate>false</task_item_duplicate>
    </task_item>
</task>
```

<task_name> indicates the screen that the navigation information is for.
<task_item_name> contains the text that is displayed on the button. This is also the key into the translation table.

<task_item_command> is the navigation command executed after the button has been processed. This may be the keywords NONE or BACK or the full classpath of another screen.

Wireless Development

Overview

The SIM Wireless handheld client is a Wavelink application. Basically 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 used via these handheld devices. In certain areas such as stock counts and product groups, some administrative tasks must be done on the PC as it cannot be performed by this 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 – Form_<name> – EventHandler_<name> – <functional area>Utility – EJB Service

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. It then receives actions from these forms back at the server – very much like a web page. These forms are not coded, but are generated from xml files (also much like web pages). A form's xml file will be screen_<*name*> and located in its own directory in the project path sim.wireless.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 actually coded by the developer to handle the logic for the handheld device.

<functional area>Utility

Quite often, 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 <functional area>Helper objects available for the service layer.

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. Here is an example of a form designed in an .xml file for entering a new item on an inventory adjustment.

Example: Screen_InventoryAdjustmentItemNew

```
<Screen name="InventoryAdjustmentItemNew">
 <LogicalScreen>
  <field name="itemId" type="string" length="21" />
  <field name="itemDesc" type="string" length="42" />
  <field name="reasonDesc" type="string" length="21" />
  <field name="unavailableQty" type="string" length="5" />
  <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">
       $[Inventory Adj.]</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="reasonDesc"
       field="reasonDesc"></label>
   <label y="6" x="0" height="1" width="10" name="unavailableLabel" />
   <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">
       $[Quantity]$[wireless.delimiter]</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>
```

Example: AbstractEventHandler_InventoryAdjustmentItemNew

```
abstract public class AbstractEventHandler_InventoryAdjustmentItemNew 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();
```

The API of the AbstractEventHandler that was created to match the form is shown in the above example. 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 to scan a value into. 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 need to be performed. There are methods for assigning data to forms, reading data from forms, displaying alerts, displaying exceptions, checking and releasing locks, showing specialized screens (barcode, yes/no choice, text input) and navigating to other forms.

YesNoEventHandler

YesNoEventHandlers are simple choice windows that display a choice and allow the user to pick 'Yes' or 'No'.

Example: RequestCancelYesNoHandler

```
public class RequestCancelYesNoHandler extends SimYesNoHandler {

  public void performYes(IApplicationForm currentForm) throws Exception {
   try {
     ItemRequest itemRequest =
            ItemRequestWirelessUtility.getContext().getItemRequest();
     if (itemRequest.getItemRequestLineItems().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 four methods above for each new YesNoHandler.

performYes() is executed when the user chooses the 'Yes' option.

performNo() is executed when the user chooses the 'No' option.

hGetTitle() returns the title to display at the top of the form.

hetMessage() 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. This context is carried in the user context 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: InventoryAdjustmentContext

```
public class InventoryAdjustmentContext {

  private InventoryAdjustment inventoryAdjustment = null;
  private UOMWrapper uom = new UOMWrapper();
  private Quantity scanEnteredQty = null;
  private Quantity computedQty = null;
  private boolean takeFromUnavailableBucket = false;

  public InventoryAdjustmentContext() {
    super();
  }

  public InventoryAdjustment getInventoryAdjustment() {
    return inventoryAdjustment;
  }

  public void setInventoryAdjustment(InventoryAdjustment inventoryAdjustment) {
    this.inventoryAdjustment = inventoryAdjustment;
  }
  // 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 UserContext.

Example: InventoryAdjustmentWirelessUtility

```
public static InventoryAdjustmentContext getContext() {
  return (InventoryAdjustmentContext)
            UserContext.getValue(InvAdjustmentWirelessKeys.CONTEXT);
}

public static InventoryAdjustmentContext createContext() {
  InventoryAdjustmentContext context = new InventoryAdjustmentContext();
  UserContext.setValue(InvAdjustmentWirelessKeys.CONTEXT, context);
  return context;
}

public static void removeContext() {
  UserContext.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 above 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 job is for the context. The example code below shows a utility method for persisting the inventory adjustment.

Example: InventoryAdjustmentWirelessUtility

```
public static void persistInventoryAdjustment(IApplicationForm currentForm) {
     InventoryAdjustmentContext context =
                         InventoryAdjustmentWirelessUtility.getContext();
        InventoryAdjustment invAdjustment = context.getInventoryAdjustment();
        try {
            if (context.getComputedQty() != null) {
                invAdjustment.setQuantity(context.getComputedQty());
            }
            if (invAdjustment.isCoherent()) {
           InventoryAdjustmentHelper.processInventoryAdjustment(invAdjustment);
                alert(currentForm,
                      InventoryAdjustmentWirelessUtility.getTitleKey(),
                      InvAdjustmentWirelessKeys.MESSAGE_ADJUSTMENT_COMPLETE,
                      ItemWirelessKeys.SCREEN_SCAN_BARCODE, false);
            }
        } catch (BusinessException be) {
            alert(currentForm,
                  InventoryAdjustmentWirelessUtility.getTitleKey(),
                  be.getLocalizedMessage(),
                  InvAdjustmentWirelessKeys.SCREEN_ITEM_NEW, 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 not be modifying this class.

Primarily, developers access this information by having an EventHandler for a form implement that PageableEventInterface. The ScreenPageManager uses this defined API to control the paging.

Example: 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 may 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: EventHandler_ItemRequestSelectRequest

```
public Map getMenuItemMap() {
  return (Map) UserContext.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 below example, 9 options (or item requests) will be displayed with order0 thru order9 assigned as their name.

Example: 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(), etc. 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: 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 very easy. In the onFormOpen() method, if there is a list available to display selections for, then retrieve the page manager and call initializeScreen(). This will setup and display the first page of selection options. It should be the last method executed in onFormOpen().

Example: EventHandler_ItemRequestSelectRequest

```
protected void onFormOpen() {
  try {
    setFormData(FIELD_INSTRUCTIONS, getLabel("Select Item Request"));

    if (getMenuItemMap() == null) {
      initializeOnEntry();
    } else {
      getPageManager().initializeScreen();
    }
  } catch (Exception e) {
     handleException(e);
  }
}
```

Coding Guidelines

The Wireless clients should always access services through the <*name*>Helper class.

Customizing Wireless Forms

This section contains some tips on customizing wireless code.

Creating a New Context

Note: Do not alter code within an existing context.

Create an entirely new context to store the custom information and place it in the UserContext. Create your own custom utility to deal with the context.

Example: MyCustomUtility

```
UserContext.getValue(MyCustomContext.ContextKey);
UserContext.setValue(MyCustomContext.ContextKey);
UserContext.removeValue(MyCustomContext.ContextKey);
```

Creating a New Utility

> **Note:** Do not alter code within an existing utility.

Create an entirely new utility to use from the eventhandlers to execute 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.

```
public class MyCustomStockCountUtility extends StockCountWirelessUtility {
```

Altering Code in an Event Handler

Because of the manner that the wavelink code functions, eventhandlers cannot be easily replaced or subclassed. Instead, the eventhandler will need to 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 an extremely 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 will need to be placed back in the custom JAR.