Oracle® Retail Store Inventory Management

Wavelink Studio Client Guide
Release 16.0
E81341-02

September 2018

ORACLE

Oracle Retail Store Inventory Management Wavelink Studio Client Guide, Release 16.0

E81341-02

Primary Author: Tracy Gunston

Value-Added Reseller (VAR) Language

Oracle Retail VAR Applications

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

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

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

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

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

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

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

Contents

Send Us Your Comments

Oracle Retail Store Inventory Management, Wavelink Studio Client Guide, Release 16.0

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

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

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

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

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

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

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

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

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

Preface

The *Oracle Retail Store Inventory Management Wavelink Studio Client* Guide contains the requirements and procedures that are necessary for the retailer to extend and customize the Store Inventory Management application.

Audience

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

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the Oracle Retail Store Inventory Management Release 16.0 documentation set:

- *Oracle Retail Store Inventory Management Implementation Batch Operations Guide*
- *Oracle Retail Store Inventory Management Implementation Configuration Guide*
- *Oracle Retail Store Inventory Management Implementation Installation Guide*
- *Oracle Retail Store Inventory Management Implementation Extension Guide*
- *Oracle Retail Store Inventory Management Implementation Integration Guide*
- *Oracle Retail Store Inventory Management Implementation MAF Guide*
- *Oracle Retail Store Inventory Management Implementation MAF Installation Guide*
- *Oracle Retail Store Inventory Management Implementation Release Notes*
- *Oracle Retail Store Inventory Management Implementation Security Guide*
- *Oracle Retail Store Inventory Management Implementation Store User Guide*
- *Oracle Retail Store Inventory Management Implementation Data Model*
- *Oracle Retail Store Inventory Management Implementation Upgrade Guide*

Customer Support

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

https://support.oracle.com

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)

- Detailed step-by-step instructions to re-create
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 16.0) or a later patch release (for example, 16.0.1). If you are installing the base release and additional patch releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch releases can contain critical information related to the base release, as well as information about code changes since the base release.

Supplemental Documentation on My Oracle Support

The following documents are available through My Oracle Support. Access My Oracle Support at the following URL:

http://support.oracle.com/

Enterprise Integration Guide (Located in the Oracle Retail Integration Suite Library on the Oracle Technology Network)

The Enterprise Integration Guide is an HTML document that summarizes Oracle Retail integration. This version of the Integration Guide is concerned with the two integration styles that implement messaging patterns: Asynchronous JMS Pub/Sub Fire-and-Forget and Web Service Request Response. The Enterprise Integration Guide addresses the Oracle Retail Integration Bus (RIB), a fully distributed integration infrastructure that uses Message Oriented Middleware (MOM) to integrate applications, and the Oracle Retail Service Backbone (RSB), a productization of a set of Web Services, ESBs and Security tools that standardize the deployment.

Improved Process for Oracle Retail Documentation Corrections

To more quickly address critical corrections to Oracle Retail documentation content, Oracle Retail documentation may be republished whenever a critical correction is needed. For critical corrections, the republication of an Oracle Retail document may at times not be attached to a numbered software release; instead, the Oracle Retail document will simply be replaced on the Oracle Technology Network Web site, or, in the case of Data Models, to the applicable My Oracle Support Documentation container where they reside.

This process will prevent delays in making critical corrections available to customers. For the customer, it means that before you begin installation, you must verify that you have the most recent version of the Oracle Retail documentation set. Oracle Retail documentation is available on the Oracle Technology Network at the following URL:

http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html

An updated version of the applicable Oracle Retail document is indicated by Oracle part number, as well as print date (month and year). An updated version uses the same part number, with a higher-numbered suffix. For example, part number E123456-02 is an updated version of a document with part number E123456-01.

If a more recent version of a document is available, that version supersedes all previous versions.

Oracle Retail Documentation on the Oracle Technology Network

Oracle Retail product documentation is available on the following web site:

http://www.oracle.com/technetwork/documentation/oracle-retail-100266.html

(Data Model documents are not available through Oracle Technology Network. You can obtain them through My Oracle Support.)

Conventions

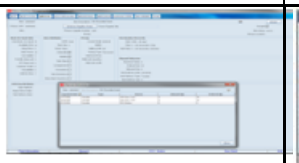
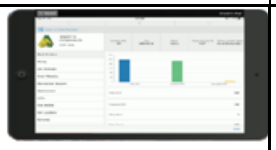
The following text conventions are used in this document:

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

1

Wavelink Studio Client

SIM UI Clients

	Desktop	Wavelink HH	MAF UI - Large	MAF UI - small
				
Technical	Java SWING	Wavelink Studio	Oracle - Mobile Application Framework	Oracle - Mobile Application Framework
Operating systems supported	Windows Desktop Version	Windows Mobile/CE, iOS, Android (Velocity client)	iOS, Android, Windows 10	iOS, Android, Windows 10
Device	PC/desktop, Tablet	Hardened Mobile devices, Tablet	Tablet	Hardened Mobile devices (MC 40/TC 70/and so on)
Used by	Store manager, Store Ops, Inventory Control	Store associates walking the shop floor	Store/district manager	Store associates walking the shop floor
User Interaction	Mouse, keyboard, Wedge Scanner	Sled scanner, touch	Bluetooth/Wedge scanner, touch	Bluetooth/Wedge scanner, touch, Barcode scanner (MC40)
Form factor - optimized	Large - Extra large (12"+)	Small (4"-5")	Medium - Large (7"+)	Small (4"-6")
Purpose	Admin, transaction approval, transaction management, research	Scanning barcodes, recording transaction information, mobility	Manager dashboard view, mobility	Scanning barcodes, recording transaction information, mobility

This document will refer to Wavelink hand-held (HH) clients' portion.

Wavelink UI Client

SIM uses Wavelink studio client as the underlying platform to render SIM UI on mobile devices and its built-in communication protocol (telnet like) to connect to server.

In addition to Windows mobile devices, Wavelink-UI has been enhanced to render on the following Wavelink Studio clients.

- iOS studio-client

The following devices were used to certify original releases. Wavelink has kept pace with iOS releases and they certify with the latest iOS versions.

  - iPod Touch - 4th Generation
    Operating System iOS 6.1.5
    Sleds Verifone PAYware e210 and Honeywell Captuvo SL22
  - iPod Touch - 5th Generation
    Operating System iOS 7.1
    Sleds Verifone PAYware e315 and Honeywell Captuvo SL42

For a more detailed look at configuration and setup and so on, please refer to the SIM Support for New Wavelink Studio Clients white paper on My Oracle Support.

- Android-velocity-Wavelink clients

The velocity android client brings all the SIM handheld capabilities to a touch enabled device running 4.4 KitKat and above. This enhanced client has more flexibility for customization.

Besides the client app which needs to be installed on the android device, there is a Velocity Console – a desktop application – which can be used to configure hosts, themes, keyboard, scanners and more. This application is available on the Wavelink website.

For more information, there is a quick start guide for configuring and setting up the Velocity client. Please refer to the Wavelink Android Quick Start Guide white paper on My Oracle Support.

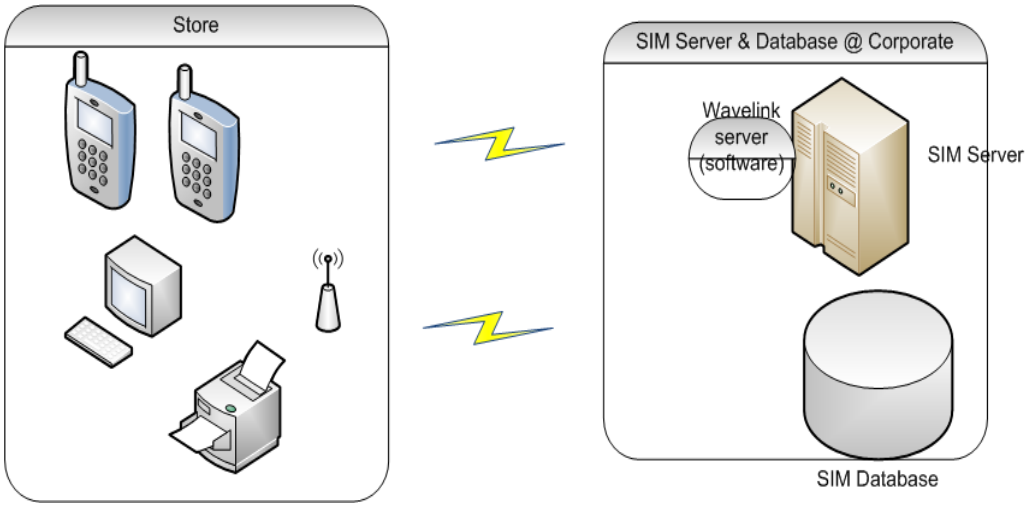
Wavelink Studio-client supports multiple wireless-devices/operating system combinations. For a complete list, please see following link:

http://www.wavelink.com/download/Main.aspx

Information on the supported sleds can be found on the LANDesk community site.

Wavelink Studio Server is bundled and installed with the SIM server. Wavelink platform allows device and operating system independence to Oracle SIM. As long as there is a Studio-client for a wireless device from Wavelink, it will render SIM UI.

The Wavelink Server is deployed centrally and serves SIM handheld clients from many stores, making calls to the SIM Server when necessary. The Wavelink Server listens for client connections on a single TCP port. The Wavelink Server is responsible for managing/load balancing each Wavelink instance.



Getting Started With Wavelink

- Download Wavelink-Studio-Client for specific Handheld device/OS from wavelink website
- Configure Handheld device with IP-address/port# of Wavelink-Studio-Server
- Wavelink-Studio-Server is designed to access SIM server through standard EJB calls
- Handheld SIM screens are rendered onto HH device
- Handheld device communicates to Wavelink-Studio-Server via telnet-like protocol

Each Wavelink-server instance can support ~150 concurrent users. It is easy to scale with load-balancer.

2

Wavelink UI Extension

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

> **Note:** SIM wireless source code has been packaged separately and can be obtained by logging a service request with Oracle Support.

Build/Packaging/Deployment Related

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

Architecture

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

For more information regarding how to gain the skills and knowledge to implement SIM successfully, see Oracle Retail's Specialization program on the Oracle Partner Network (OPN) located here:

http://www.oracle.com/partners/en/partner-with-oracle/training/index.html#cw01step-2.htm

Building Wireless Forms

The SIM Wireless handheld client is a Wavelink application. All wireless clients (users in the store) use handheld physical devices to talk to a wireless container (sometimes called the wireless server). Most of the SIM application can be accessed through these handheld devices. In certain areas such as stock counts and product groups, some administrative tasks must be done on the PC as these tasks cannot be performed by the handheld device.

Wireless Application Architecture

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

Wireless Framework

This is the wireless framework application consisting of the Wavelink code that starts and executes as a server and handles the actual communication protocol back and forth to the handheld devices.

Form_<name>

The framework loads and displays forms on the handheld device. The framework then receives actions from these forms back at the server – similar to a Web page. These forms are not coded, but are generated from xml files (also similar to Web pages). A form's xml file will be named **screen_<name>** and located in its own directory in the wireless project with the path `generator.screens`.

EventHandler_<name>

AbstractEventHandlers are generated along with the forms to receive actions. The EventHandler_<name> class extends the abstract class and actually implements the actions that can be received from the form. The EventHandler classes are coded by the developer to handle the logic for the handheld device.

<functional area>Utility

Often, an EventHandler communicates with the rest of the SIM system by using logic available through a utility that covers common logic within a functional area.

EJB Client

The EJBs to the regular SIM services are accessed from EventHandlers or Utilities by using the client-side service client (either by specifically instantiating or by using ClientServiceFactory).

Forms

Forms contain the page information that is sent back and forth to the actual device.

Event Handler

When a form is created, an AbstractEventHandler_<name>.java file must exist as well. The developer must then code an EventHandler_<name>.java file that matches the abstract handler. This section will outline how an EventHandler relates to a form and some of the general methods that are available to an EventHandler. The *Example: Screen_InventoryAdjustmentNormalItem* is an example of a form designed in an .xml file for entering an item on an inventory adjustment:

Example: Screen_InventoryAdjustmentNormalItem

```
<Screen name="InventoryAdjustmentNormalItem">
  <LogicalScreen>
    <field name="itemId" type="string" length="21" />
    <field name="itemDesc" type="string" length="42" />
    <field name="reason" type="string" length="21" />
    <field name="unavailableLabel" type="string" length="10" />
    <field name="unavailableQty" type="string" length="5" />
    <field name="qtyLabel" type="string" length="15" />
    <field name="qty" type="string" length="5">1</field>
    <field name="uom" type="string" length="5" />
    <field name="packSize" type="string" length="5" />
  </LogicalScreen>
<PhysicalScreens deviceclass="dnw">
    <PhysicalScreen seq="0">
      <label y="0" x="0" height="1" width="21"
            style=".heading1">$[wireless.inventoryAdjustment]</label>
      <label y="2"  x="0"  height="1" width="21" name="itemId"
field="itemId"></label>
      <label y="3"  x="0"  height="2" width="21" name="itemDesc"
field="itemDesc"></label>
      <label y="5"  x="0"  height="1" width="21" name="reason"
field="reason"></label>

      <label y="6"  x="0"  height="1" width="10" name="unavailableLabel"
             field="unavailableLabel"></label>
      <label y="6"  x="10"  height="1" width="6" name="unavailableQty"
             field="unavailableQty"></label>
      <label y="6"  x="16"  height="1" width="5" name="unavailableUnits" />

      <label y="8"  x="0" height="1" width="9" name="qtyLabel"
field="qtyLabel"></label>
      <input y="8"  x="10" height="1" width="6"  name="qty" field="qty" seq="0"
             acceptScan="false" validateKeyEventOnScan="false"/>
      <label y="8"  x="16" height="1" width="5"  name="uom" field="uom" />

      <label y="9"  x="0"  height="1" width="10" name="packSizeLabel" />
      <input y="9"  x="10" height="1" width="6" name="packSize" field="packSize"
             seq="1" acceptScan="false" validateKeyEventOnScan="false"/>

      <scan/>

      <cmdkey y="0" x="0" width="0" height="0" key="&toggle;" name="toggle"
            action="callMethod" target="doToggle"/>
     <cmdkey y="0" x="0" width="0" height="0" key="&exit;" name="Exit"
           action="callMethod" target="doExit" />
  </PhysicalScreen>
</PhysicalScreens>
</Screen>
```

The API of the AbstractEventHandler that was created to match the form is shown in the example below:

Example: AbstractEventHandler_InventoryAdjustmentNormalItem

```
abstract public class AbstractEventHandler_InventoryAdjustmentNormalItem extends SimEventHandler {
    abstract protected void onFormOpen();
    abstract protected void onFormClose();
    abstract public void onScan(String data);
    abstract public boolean qty_OnChange(String newValue);
    abstract public boolean qty_OnExit(String newValue);
    abstract public boolean packSize_OnChange(String newValue);
    abstract public boolean packSize_OnExit(String newValue);
```

```
    abstract public void doToggle();
    abstract public void doExit();
}
```

OnFormOpen() and OnFormClose() must exist for each AbstractEventHandler regardless of the form:

- OnFormOpen() is executed before the form is displayed to the handheld device. The code that populates the form with data should be placed in this method.
- OnFormClose() is executed when the form is removed from the handheld device.

The onScan() method matches the <scan/> tag in the xml, which indicated a row on the form into which to scan a value. When a value is scanned by the device, the information is passed to the onScan() method as a data parameter. The developer can implement this method in the EventHandler to process the scanned data (in this case, the barcode of an item).

Both qty_OnChange() and qty_OnExit() were created by the <input> tag with the field=qty set within that tag. Since qty was entered as the field, these two methods exist to handle processing when a quantity is entered.

Both packSize_OnChange() and packSize_OnExit() were created by the <input> tag with the field =packSize set within that tag. Since packSize was entered as the field, these two methods exist to handle processing when a pack size is entered.

The doToggle() method was created by the <cmdkey> tag based on the target=value section of the tag. Since doToggle was entered as the target value, this method was created. A cmdkey displays as an option on the screen. The method is executed in the EventHandler when the option is chosen on the screen. This is the same with the doExit() based on its <cmdkey> tag.

SimEventHandler

Each AbstractEventHandler extends from SimEventHandler, a superclass that contains methods for common tasks. These methods should always be used when these tasks must be performed. There are methods for the following:

- Assigning data to forms
- Reading data from forms
- Displaying alerts
- Displaying exceptions
- Checking locks
- Releasing locks
- Showing specialized screens (barcode, Yes/No choice, text input)
- Navigating to other forms

YesNoEventHandler

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

Example: RequestCancelYesNoHandler

```
public class RequestCancelYesNoHandler extends SimYesNoHandler {
  public void performYes(IApplicationForm currentForm) throws Exception {
    try {
      ItemRequest itemRequest = ItemRequestWirelessUtility.getContext().getItemRequest();
      if (itemRequest.getLineItems().size() > 0) {
          ItemRequestWirelessUtility.doSave(itemRequest, false);
      }
```

```
      currentForm.gotoForm(ItemRequestWirelessKeys.SCREEN_MENU);
       } catch (BusinessException be) {
           currentForm.gotoForm(ItemRequestWirelessKeys.SCREEN_MENU);
       } catch (Exception e) {
           handleException(e, currentForm);
       }
  }
  public void performNo(IApplicationForm currentForm) throws Exception {
    try {
        currentForm.gotoForm(ItemRequestWirelessKeys.SCREEN_SUMMARY);
    } catch (Exception e) {
        handleException(e, currentForm);
    }
  }
  public String getTitle() throws Exception {
    return getText(ItemRequestWirelessUtility.getTitleKey());
  }
  public String getMessage() throws Exception {
    String id = ItemRequestWirelessUtility.getContext().getItemRequest().getId();
    return getMessage(ItemRequestWirelessKeys.MESSAGE_EXIT_CONFIRM, id);
  }
}
```

The SimYesNoHandler superclass that all YesNoHandlers should extend contains most of the same helper methods as the SimEventHandler class. This means that such functionality as translating text and handling exceptions should always use these helper methods.

The SimYesNoHandler also has the implemented code that returns the Yes and No choice labels for the screen, so the user only has to implement the following four methods for each new YesNoHandler:

- performYes() is executed when the user chooses the **Yes** option.
- performNo() is executed when the user chooses the **No** option.
- getTitle() returns the title to display at the top of the form.
- getMessage() return the query text to display on the form.

Wireless Context

A context represents a repository of data entered or being altered for a particular functional area within the user session. This context is carried in the repository to make it readily accessible between different forms. The InventoryAdjustmentContext is used as an example to trace some of the usages of context. Note that the context object itself simply has a set of data variables.

Example: InventoryAdjustmentContext

```
public class InventoryAdjustmentContext {
    private InventoryAdjustment inventoryAdjustment;
    private InventoryAdjustmentReason lastAdjustmentReason = null;
    private String lastScannedUIN = null;
    private Quantity scanEnteredQty = null;
    private Quantity computedQty = null;
    private boolean takeFromUnavailableBucket = false;
    private int currentUINIndex = 0;
    public InventoryAdjustment getInventoryAdjustment() {
        return inventoryAdjustment;
    }
    public void setInventoryAdjustment(InventoryAdjustment inventoryAdjustment)     {
        this.inventoryAdjustment = inventoryAdjustment;
    }
    public boolean isTakeFromUnavailableBucket() {
```

```
        return takeFromUnavailableBucket;
    }
    public void setTakeFromUnavailableBucket(boolean takeFromUnavailableBucket)     {
        this.takeFromUnavailableBucket = takeFromUnavailableBucket;
    }
    // Other Getters & Setters
}
```

Access to the context is through the utility for the functional area, which contains a set of static helper methods to create, retrieve and remove the context. Note that the actual context itself is stored within the SimWirelessRepository.

Example: InventoryAdjustmentWirelessUtility

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

public static InventoryAdjustmentContext createContext() {
    InventoryAdjustmentContext context = new InventoryAdjustmentContext();
    SimWirelessRepository.setValue(InvAdjustmentWirelessKeys.CONTEXT, context);
    return context;
}
public static void removeContext() {
   SimWirelessRepository.removeValue(InvAdjustmentWirelessKeys.CONTEXT);
}
```

Wireless Utilities

Wireless Utilities are static classes that contain numerous helper methods to execute business logic, such as in the examples for context in which the context is created or retrieved, or in the following example where a new inventory adjustment was created. There is usually only one <name>WirelessUtility for each functional workflow on the handheld device, but some large areas may have more than one.

The Wireless Utility contains public static helper methods to perform business logic for EventHandlers. Wireless Utilities do not contain any data variables at the class level as the utility does not store state of its data. That responsibility is handled by the context. Example: InventoryAdjustmentWirelessUtility. The below example code shows a utility method for persisting the inventory adjustment:

Example: InventoryAdjustmentWirelessUtility

```
public static void persistInventoryAdjustment(IApplicationForm currentForm) {
    InventoryAdjustment invAdjustment = getContext().getInventoryAdjustment();
    try {
        if (invAdjustment.isCoherent()) {
            ClientServiceFactory.getInventoryAdjustmentServices()
                .updateInventoryAdjustment(invAdjustment);
        alert(currentForm, getTitleKey(), InvAdjustmentWirelessKeys.MESSAGE_ADJUSTMENT_COMPLETE,
               ItemWirelessKeys.SCREEN_SCAN_BARCODE, false);
        }
     } catch (BusinessException businessException) {
        String message = getMessage(businessException.getMessage(),
                                    businessException.getParameters());
        alert(currentForm, getTitleKey(), message, getItemScreen(), true);
     } catch (Exception e) {
        handleException(e, currentForm);
     }
}
```

Form Paging

The ScreenPageManager and PageableEventInterface are used to create selection screens that go on for more than one visual screen worth of information on the handheld device. A **next** and **previous** command option exists on the base of the form to scroll through the pages.

The ScreenPageManager is a wireless framework class that handles a lot of the formatting and displaying of the options on a form and controls going forward and backward through the pages. Developers should never modify this class.

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

Example: PageableEventInterface

```
public abstract Map getMenuItemMap();
public abstract String getScreenName();
public abstract IApplicationForm getCurrentForm();
public abstract String[] getPageFieldNames();
public abstract String[] getSpecialFieldValues();
```

The ItemRequestSelectRequest form displays a list of requests for the user to select from. These requests might not all fit on one form, so this EventHandler implemented 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) SimWirelessRepository.getValue(ItemRequestWirelessKeys.ORDER_MAP);
}
public String getScreenName() {
    return ItemRequestWirelessKeys.SCREEN_SELECT_REQUEST;
}
```

The method getCurrentForm() is implemented by SimEventHandler, so all EventHandlers automatically implement that method. The getPageFieldNames() method returns a list of names to assign to the options displayed on the screen. So in the following example, nine options (or item requests) are displayed with **order0** through **order9** assigned as their name:

Example: EventHandler_ItemRequestSelectRequest

```
private static final String[] orderFieldNames = { "order0", "order1", "order2",  "order3",
"order4", "order5", "order6", "order7", "order8", "order9" };

 public String[] getPageFieldNames() {
    return orderFieldNames;
}

 public String[] getSpecialFieldValues() {
    return null;
}
```

When the option is selected, the Wavelink framework calls back to a method named after the options names assigned in getPageFieldNames(). In this case, selectOrder0(), selectOrder1(), and so on. This is where the developer can place code to handle the selection. There are no examples of getSpecialFieldValues(). This method returns null in all EventHandlers that implement the interface.

Example: EventHandler_ItemRequestSelectRequest

```
public void selectOrder0() {
    selectOrder(0);
}
public void selectOrder1() {
    selectOrder(1);
}
// Rest of orderFieldNames…public void selectOrder9() {
    selectOrder(9);
}
```

Once the methods are defined, starting the page display is easy. In the onFormOpen() method, if there is a list available to display selections for, then retrieve the page manager and call initializeScreen(). This sets up and displays the first page of selection options. It should be the last method executed in onFormOpen():

Example: EventHandler_ItemRequestSelectRequest

```
protected void onFormOpen() {
  try {
      LogService.debug(this, getClass().getSimpleName() + ".onFormOpen()");
      setFormText(FIELD_INSTRUCTIONS, getLabel("Select Item Request"));
      if (getMenuItemMap() == null || getPageManager() == null) {
          initializeOnEntry();
      } else {
          getPageManager().initializeScreen();
      }
  } catch (Exception e) {
```

```
      handleException(e);
  }
}
```

Customizing Wireless Forms

This section contains some tips on customizing wireless code. Customizing wireless code requires modifying source code in SIM. This needs to be handled with care.

Coding Guidelines

- The wireless clients should always access services through the ClientServiceFactory class.
- Do not alter existing context classes. Create new ones instead and store in SimWirelessRepository.
- When at all possible, design the modifications to be new forms or replacements to current forms instead of modifying current forms.
- Similar to forms/screens, attempt to write new utility classes rather than modifying existing ones.

By following the guidelines above, the only code that should need to be modified in SIM is the code that directs one form to a newly designed form – keeping the amount of rework at a minimum for future releases.

Creating a New Context

If you need to store additional state data during a session, create an entirely new context to store the information. The newly created context must then be placed in the SimWirelessRepository. Next, create a custom wireless utility to handle interaction with the context, including placing the context in the SimWirelessRepository.

```
public class MyCustomContext() {
    { . . . . } // Context code.
 }
```

Creating a New Utility

If new functionality is desired in the wireless application that needs to access a new context or access new services, then the developer should create an entirely new utility to use from the eventhandlers to execute this code.

If you need ready access to a utility method or need to override the functionality of a utility method, then subclass the utility in question with your new utility as shown in Example: MyCustomUtility.

Example: MyCustomUtility

```
public class MyCustomStockCountUtility extends StockCountWirelessUtility {
    public static MyCustomContext getContext() {
      return (MyCustomContext) SimWirelessRepository.getValue(MyCustomContextKeys.CONTEXT);
    }
    public static MyCustomContext createContext() {
        MyCustomContext context = new MyCustomContext ();
        SimWirelessRepository.setValue(MyCustomContextKeys.CONTEXT, context);
        return context;
    }
    public static void removeContext() {
       SimWirelessRepository.removeValue(MyCustomContextKeys.CONTEXT);
    }
    [ . . . . ] // New Utility Code
}
```

Altering Code in an EventHandler

Because of the manner that the Wavelink code functions, eventhandlers cannot be easily replaced or sub-classed. Instead, the eventhandler must be removed from the JAR, re-coded, and placed in a custom JAR earlier in the classpath. These modifications are not guaranteed to function correctly with newer releases.

Altering Code in a Form

Altering a Wavelink form is a complicated process:

1. Alter the form_<name> source code.
2. Alter the form_dnw_<name> source code.
3. Alter the AbstractEventHandler_<name> source code.
4. Alter the EventHandler_<name> source code.

All of these files must be placed back in the custom JAR that is first in the classpath.

Significant Classes to Understand

Before attempting to modify or create new handheld Wavelink forms, read all documentation supplied by Wavelink. In addition, the following is a list of critical framework classes specific to SIM:

- CommandListHelper
- CommandListInterface
- InputTextInterface
- LocaleWirelessUtility
- PageableEventInterface
- PrintSelectInterface
- ScreenPageManager
- SimBasicHandler
- SimEventHandler
- SimWirelessRepository
- SimYesNoHandler
- WirelessExceptionManager
- WirelessPrintService
- WirelessUtility
- WirelessValidation
- YesNoEventInterface

Wireless Internationalization

Wireless Internationalization is not easily customizable. Many of the previous techniques described will automatically update the same content on the HH forms, but that is not always the case. The following documentation describes how to code HH forms to take advantage of the built in localization that exists in SIM. Wireless customization requires original source code.

Internationalization is handled by different approaches based on where in the Wireless code the translation is needed.

Forms

In the xml files that define the handheld forms, the display of labels is usually surrounded with a $[ ] indicator. In the Form<name>.java classes, the text within the brackets is wrapped by an AppGlobal.getString() call which translates the text.

Example: 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 event handlers.

Key methods include:

SetFormDate()

Formats a date for the locale and country and places it in the form.

SetFormInteger()

Formats an integer for the locale and places it in the form.

SetFormDecimal()

Formats a decimal for the locale and places it in the form.

SetFormQuantity()

Formats a quantity for the locale and places it in the form.

GetText()

Retrieves the translation of the text.

GetLabel()

Retrieves the translation of the text followed by the label delimiter

GetMessage()

Retrieves the translation of the message

GetFormInteger()

Retrieves entered text as an integer

GetFormDecimal()

Retrieves entered text as a decimal

GetFormQuantity()

Retrieves entered text as a quantity

HandleException()

Handles displaying an exception (translating the message).

Here are some examples of standard eventhandler code using these internationalization methods. In the first example, we are translating the label **Select PO From**.

Example: 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

```
EventHandler_ContainerLookupDetail
    protected void onFormOpen() {
        try {
            LogService.debug(this, this.getClass().getSimpleName() + ".onFormOpen()");
            ShipmentCartonVO cartonVO = (ShipmentCartonVO)
SimWirelessRepository.getValue(ContainerWirelessKeys.USER_CONTEXT_CONTAINER);
            SourceVO fromLocation = cartonVO.getFromLocation();

            setFormText("containerId", cartonVO.getCartonId());
            setFormText("status", getText(cartonVO.getStatus().toString()));
            setFormText("fromLocation", fromLocation.getId() + " " + fromLocation.getName());
            setFormText("asnNumber", cartonVO.getAsnId());
            setFormDate("eta", cartonVO.getEta());
```

```
            setFormDate("receiptDate", cartonVO.getReceiveDate());
            setFormText("receiptTime",
LocaleWirelessUtility.formatTime(cartonVO.getReceiveDate()));
            setFormInteger("totalCases", cartonVO.getNumberOfCasesExpected());
        } catch (Exception e) {
            handleException(e);
        }
    }
```

<name>WirelessUtility

Every wireless utility class extends from WirelessUtility, which like SimEventHandler, contains numerous methods to assist in formatting and retrieving information in an internationalized manner. These helper methods in the superclass should always be used to perform these types of tasks when coding utility methods. Read the javadoc on WirelessUtility for complete descriptions.

Key methods include:

GetText()

Retrieves the translation of the text.

Alert()

Handles displaying an alert message (translating the message).

GetLabel()

Retrieves the translation of the text followed by the label delimiter.

GetMessage()

Retrieves the translation of the message.

HandleException()

Handles displaying an exception (translating the message).

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

Example: StockCountWirelessUtility

```
public static String getDescription(StockCount stockCount) {
    StringBuilder buffer = new StringBuilder();
    try {
        if (stockCount.getType() == StockCountType.PROBLEM_LINE) {
            buffer.append(getLabel("wireless.problemLineABBV"));
        }
        String text = stockCount.getCountDescription().trim();
        int index = text.lastIndexOf("(");
        if (index < 0) {
            buffer.append(text);
        } else {
            buffer.append(text.substring(index));
            buffer.append(StringConstants.SPACE);
            buffer.append(text.substring(0, index).trim());
        }
        return buffer.toString();
    } catch (Exception e) {
        return getText("wireless.noDescABBV");
    }
}
```

In Example: *StockCountWirelessUtilit*y business logic is required to create a stock count description. The utility uses the getLabel() and getText() helper methods to guarantee that the text is translated as the description is being built.

LocaleWirelessUtility

If there are no superclass helper methods available for what you want to accomplish, you can directly use the LocaleWirelessUtility to perform internationalized functions.

Wireless Labels

Wireless labels have an additional consideration that is not needed on the PC. The width of a wireless form, which is displayed on a handheld device, is very narrow. That means only a small amount of space is allocated to a label. The English labels used as translation keys are defined by the space they take up. Oracle Retail suggests that instead of using a standard English label such as **status** for the key, use **wireless.status** instead, so that it is clear in the GUI administrative screen that the translation being supplied is for the wireless device.