Sterling Commerce
An IBM Company

Sterling Supply Chain Applications

Customization Guide

Release 7.9

October 2006

THIRD PARTY SOFTWARE

THE ECLIPSE FOUNDATION SOFTWARE

The Sterling Commerce Software is also distributed with or on the same storage media as the following software com.ibm.icu_3.4.4.1.jar, org.eclipse.core.commands_3.2.0.I20060605-1400.jar, org.eclipse.core.contenttype_3.2.0.v20060603.jar, org.eclipse.core.expressions_3.2.0.v20060605-1400.jar, org.eclipse.core.filesystem.linux.x86_1.0.0.v20060603.jar, org.eclipse.core.filesystem_1.0.0.v20060603.jar, org.eclipse.core.jobs_3.2.0.v20060603.jar, org.eclipse.core.runtime.compatibility.auth_3.2.0.v20060601.jar, org.eclipse.core.runtime_3.2.0.v20060603.jar, org.eclipse.equinox.common_3.2.0.v20060603.jar, org.eclipse.equinox.preferences_3.2.0.v20060601.jar, org.eclipse.equinox.registry_3.2.0.v20060601.jar, org.eclipse.help_3.2.0.v20060602.jar, org.eclipse.jface.text_3.2.0.v20060605-1400.jar, org.eclipse.jface_3.2.0.I20060605-1400.jar, org.eclipse.osgi_3.2.0.v20060601.jar, org.eclipse.swt.gtk.linux.x86_3.2.0.v3232m.jar, org.eclipse.swt_3.2.0.v3232o.jar, org.eclipse.text_3.2.0.v20060605-1400.jar, org.eclipse.ui.workbench.texteditor_3.2.0.v20060605-1400.jar, org.eclipse.ui.workbench_3.2.0.I20060605-1400.jar, org.eclipse.ui_3.2.0.I20060605-1400.jar (collectively, Eclipse Software). All Eclipse Software is distributed under the terms and conditions of the Eclipse Foundation Software User Agreement and/or terms and conditions of the Eclipse Public License Version 1.0 (EPL) or other license agreements, notices or terms and conditions referenced for the individual pieces of the Eclipse Software. A copy of the Eclipse Foundation Software User Agreement is found at <install_dir>/rcpdependencies/windows/eclipse/plugins/notice.html and <install_dir>/rcpdependencies/gtk.linux_x86/eclipse/plugins/notice.html. A copy of the EPL is found at <install_dir>/rcpdependencies/windows/eclipse/plugins/epl-v10.html and <install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/epl-v10.html. The references to the license agreements, notices or terms and conditions governing each individual piece of the Eclipse Software are found in the following directory files for the individual pieces of the Eclipse Software:

<install_dir>/rcpdependencies/windows/eclipse/plugins/about_com.ibm.icu_3.4.4.1.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.commands_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.contenttype_3.2.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.expressions_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.filesystem.win32.x86_1.0.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.filesystem_1.0.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.jobs_3.2.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.runtime.compatibility.auth_3.2.0.v20060601.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.runtime.compatibility.registry_3.2.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.core.runtime_3.2.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.equinox.common_3.2.0.v20060603.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.equinox.preferences_3.2.0.v20060601.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.equinox.registry_3.2.0.v20060601.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.help_3.2.0.v20060602.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.jface.text_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.jface_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.osgi_3.2.0.v20060601.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.swt.win32.win32.x86_3.2.0.v3232m.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.swt_3.2.0.v3232o.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.text_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.ui.workbench.texteditor_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.ui.workbench_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/windows/eclipse/plugins/about_org.eclipse.ui_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_com.ibm.icu_3.4.4.1.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.commands_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.contenttype_3.2.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.expressions_3.2.0.v20060605-1400.html

<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.filesystem.win32.x86_1.0.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.filesystem_1.0.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.jobs_3.2.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.runtime.compatibility.auth_3.2.0.v20060601.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.runtime.compatibility.registry_3.2.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.core.runtime_3.2.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.equinox.common_3.2.0.v20060603.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.equinox.preferences_3.2.0.v20060601.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.equinox.registry_3.2.0.v20060601.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.help_3.2.0.v20060602.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.jface.text_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.jface_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.osgi_3.2.0.v20060601.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.swt.win32.win32.x86_3.2.0.v3232m.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.swt_3.2.0.v3232o.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.text_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.ui.workbench.texteditor_3.2.0.v20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.ui.workbench_3.2.0.I20060605-1400.html
<install_dir>/rcpdependencies/gtk.linux.x86/eclipse/plugins/about_org.eclipse.ui_3.2.0.I20060605-1400.html

These licenses only apply to the Eclipse Software and do not apply to the Sterling Commerce Software, or any other Third Party Software.

The org.eclipse.core.runtime_3.2.0.v20060603.jar piece of the Eclipse Software was modified slightly in order to remove classes containing encryption items. The org.eclipse.core.runtime_3.1.0.jar was modified to remove the Cipher, CipherInputStream and CipherOutputStream classes and rebuild the org.eclipse.core.runtime_3.2.0.v20060603.jar.

WARRANTY DISCLAIMER

This documentation and the Sterling Commerce Software which it describes are licensed either "AS IS" or with a limited warranty, as set forth in the Sterling Commerce license agreement. Other than any limited warranties provided, NO OTHER WARRANTY IS EXPRESSED AND NONE SHALL BE IMPLIED, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR USE OR FOR A PARTICULAR PURPOSE. The applicable Sterling Commerce entity reserves the right to revise this publication from time to time and to make changes in the content hereof without the obligation to notify any person or entity of such revisions or changes.

The Third Party Software is provided 'AS IS' WITHOUT ANY WARRANTY AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. FURTHER, IF YOU ARE LOCATED OR ACCESSING THIS SOFTWARE IN THE UNITED STATES, ANY EXPRESS OR IMPLIED WARRANTY REGARDING TITLE OR NON-INFRINGEMENT ARE DISCLAIMED.

Without limiting the foregoing, the Eclipse Software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Contents

3   Customizing the Console JSP Interface

4   Customizing the Configurator Swing Interface

5   Creating Custom Mobile User Interfaces

6   Customizing the Sterling Rich Client Platform Interface

7 Extending the Sterling Supply Chain Applications Database

8   Programming Transactions

B   RCP General Concepts Reference

C Console JSP Interface Extensibility Reference

D   Mobile User Interface Extensibility Reference

Index

Preface

This manual explains how to develop enterprise-wide customizations to the Sterling Supply Chain Applications.

Intended Audience

This manual is intended for use by those who are responsible for customizing Sterling Supply Chain Applications

Structure

Each chapter in this manual is organized in the same sequence that a user navigates through the user interface. For example, a chapter first describes how to modify the Sign In screen before it describes how to modify any other screens the user sees at a later point. However, you do not need to read the topics in sequential order, since each topic describes a separate, self-contained task. Each topic is organized in the following manner:

- Explanation of concept
- Description of default behavior
- Information about what you can and cannot modify
- Step-by-step procedures

This manual contains the following sections:

Chapter 1, "Introduction"

This chapter discusses extensibility in general terms and provides an overview of the Sterling Service Definition Framework and Sterling Presentation Framework.

Chapter 2, "Getting Started"

This chapter explains how to prepare your environment to perform customizations to the UI and database.

Chapter 3, "Customizing the Console JSP Interface"

This chapter walks you through the tasks required for customizing the Application Consoles User Interface.

Chapter 4, "Customizing the Configurator Swing Interface"

This chapter describes the Configurator Java Swing user interface and walks you through the tasks necessary for customizing it to suit your business needs.

Chapter 5, "Creating Custom Mobile User Interfaces"

This chapter describes the concepts and procedures for creating custom mobile device user interface applications.

Chapter 6, "Customizing the Sterling Rich Client Platform Interface"

This chapter describes the Sterling Rich Client Platform and walks you through the tasks necessary for customizing the Sterling Rich Client Platform user interface to suit your business needs.

Chapter 7, "Extending the Sterling Supply Chain Applications Database"

This chapter explains how to extend tables thar are part of Sterling Supply Chain Applications in order to maintain more attributes.

Chapter 8, "Programming Transactions"

This chapter explains how to invoke Sterling Supply Chain Applications and extended APIs and how to program user exits and event handlers.

Appendix A, "Special Characters Reference"

This appendix contains reference material and guidelines for using special characters in Sterling Supply Chain Applications.

Appendix B, "RCP General Concepts Reference"

This appendix contains reference material for understanding the general concepts of RCP.

Appendix C, "Console JSP Interface Extensibility Reference"

This appendix contains reference material that enables you to customize the user interface.

Appendix D, "Mobile User Interface Extensibility Reference"

This appendix contains the supporting reference material necessary to customize mobile device user interfaces.

Sterling Supply Chain Applications Documentation

For more information about the Sterling Supply Chain Applications Platform® components, see the following manuals in the Sterling Supply Chain Applications® documentation set:

- *Sterling Supply Chain Applications® Release Notes*

- *Sterling Supply Chain Applications® Installation Guide*

- *Sterling Supply Chain Applications® Upgrade Guide*

- *Sterling Supply Chain Applications® Performance Management Guide*

- *Sterling Supply Chain Applications® High Availability Guide*

- *Sterling Supply Chain Applications® System Management Guide*

- *Sterling Supply Chain Applications® Localization Guide*

- *Sterling Supply Chain Applications® Customization Guide*

- *Sterling Supply Chain Applications® Integration Guide*

- *Sterling Supply Chain Applications® Product Concepts*

- *Sterling Warehouse Management System® Concepts Guide*

- *Sterling Supply Chain Applications Platform® Configuration Guide*

- *Sterling Distributed Order Management® Configuration Guide*

- *Sterling Supply Collaboration® Configuration Guide*

- *Sterling Global Inventory Visibility® Configuration Guide*

- *Sterling Product Management® Configuration Guide*

- *Sterling Logistics Management® Configuration Guide*

- *Sterling Reverse Logistics® Configuration Guide*

- *Sterling Warehouse Management System® Configuration Guide*

- *Sterling Supply Chain Applications Platform® User Guide*

- *Sterling Distributed Order Management® User Guide*

- *Sterling Supply Collaboration® User Guide*

- *Sterling Global Inventory Visibility® User Guide*

- *Sterling Logistics Management® User Guide*

- *Sterling Reverse Logistics® User Guide*

- *Sterling Warehouse Management System® User Guide*

- *Sterling Supply Chain Mobile Application® User Guide*

- *Sterling Supply Chain Analytics® Guide*

- *Sterling Supply Chain Applications® Javadocs*

- *Sterling Supply Chain Applications® Glossary*

- *Sterling Parcel Carrier Adapter® Guide*

- *Sterling Application Server® Installation Guide* (for optional component)

Conventions

The following conventions may be used in this manual:

Convention	Meaning
. . .	An ellipsis represents information that has been omitted.

Convention	Meaning
< >	Angle brackets indicate user-supplied input.
`mono-spaced text`	Mono-spaced text indicates a file name, directory path, attribute name, or an inline code example or command.
/ or \	Slashes and backslashes are file separators for Windows, UNIX and LINUX operating systems. The file separator for the Windows operating system is "\" and the file separator for Unix and Linux systems is "/". The Unix convention is used unless otherwise mentioned.
<YANTRA_HOME>	User-supplied location of the Sterling Supply Chain Applications installation directory.
<YFS_HOME>	Location of the generated <YANTRA_ HOME>/Runtime directory.
<YANTRA_HOME_OLD>	User-supplied location of the Sterling Supply Chain Applications installation directory for previously installed releases. This is only applicable for Release 7.7 or above.
<YFS_HOME_OLD>	This is the <YANTRA_HOME_OLD>/Runtime directory of previously installed releases.

1

Introduction

This manual describes how to customize Sterling Supply Chain Applications. Sterling Supply Chain Applications contain the following components that can be extended to meet your business needs:

- Sterling Supply Chain Application Consoles

- Sterling Supply Chain Applications Configurator

- Sterling Supply Chain Mobile Application

1.1 Sterling Supply Chain Applications Extensibility

This chapter introduces the types of extensibility possible through Sterling Supply Chain Applications. It provides an overview, describes high-level concepts, and provides technical architectural diagrams of Sterling Supply Chain Applications.

Look and Feel Extensibility

Sterling Supply Chain Applications provide a Presentation Framework toolkit that enables you to change the way information is rendered or displayed, without changing the way it functions.

Transactional Extensibility

Sterling Supply Chain Applications provide a Service Definition Framework, which is an infrastructure that automates the conversion and transportation of data between Sterling Supply Chain Applications and third-party applications, and then converts that data into formats readable by each system. The Service Definition Framework also handles

logging and exceptions. It enables you to create custom transactions that extend the functionality of Sterling Supply Chain Applications.

Database Extensibility

In addition to customizing the user interface and transactions, you can extend the database to store additional attributes specific for your business.

Printed Documents Extensibility

You can customize printed documents. For example, you can extend the default length of bar codes.

You can use JasperReports for generating printed reports in Sterling Supply Chain Applications. Additionally, you can also generate reports as a PDF object in Sterling Supply Chain Applications. The JasperReports installation guidelines are provided in the `<YANTRA_ HOME>/Applications/Foundation/documentation/code_ examples/jasperreports/readme.html` file. A sample JasperReport called sampleOrderReport.pdf is also available in the same directory.

A Jasper print component is defined in the Service Definition Framework which can be used to automatically print a document based on an event. It is a standard XML-based component that accepts XML as input and provides an output XML. For more information on this component and also the print transaction refer to the *Sterling Supply Chain Applications Platform Configuration Guide*.

Browser Portal Extensibility

You can export views into Sterling Supply Chain Applications database that list a user's frequently used search criteria.

1.2 The Sterling Supply Chain Application Consoles User Interface

The Sterling Supply Chain Application Consoles is the user interface for conducting and tracking day-to-day transactional business, such as orders and inventory. The Sterling Supply Chain Application Consoles is also known by the terms Application Consoles or console.

The Application Consoles UI uses HTML within Java Server Pages (JSPs). The user interface layer accesses the exposed APIs through services defined in the Service Definition Framework, which ensures that only exposed APIs are used.

The UI layer of the Service Definition Framework uses very minimal XML manipulation. Wherever significant manipulation of XML output becomes necessary, changes to the APIs provide a more UI friendly output.

Figure 1–1 shows the technical architecture of the Application Consoles User Interface.

Figure 1–1   Application Consoles User Interface Architecture
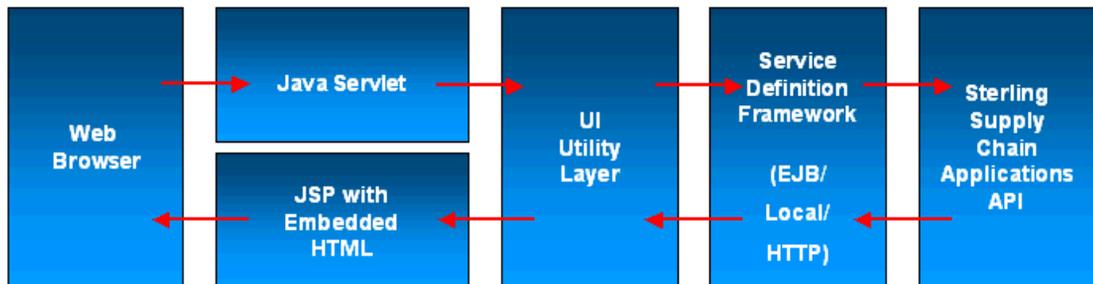
This user interface can be extended. For detailed information about extending this user interface, see the following chapters:

- Chapter 3, "Customizing the Console JSP Interface"
- Appendix C, "Console JSP Interface Extensibility Reference"

1.3 The Sterling Supply Chain Applications Configurator User Interface

The Sterling Supply Chain Applications Configurator is the user interface for configuring the setup of an organization's business rules and transactions. The Configurator is composed of Java Swing pages.

For detailed information about extending the Configurator user interface, see Chapter 4, "Customizing the Configurator Swing Interface".

1.4 The Sterling Supply Chain Applications Mobile User Interface

Sterling Supply Chain Applications enable you to develop and display a custom user interface for mobile devices used in warehouse operations.

The mobile architecture consists of two components: a client and a server.

- Sterling Supply Chain Applications mobile client - typically a Pocket PC-based handheld device, but it can also be a VT220 emulation terminal. In this documentation, the mobile client is referred to as a *mobile device*.

- application server - Sterling Supply Chain Applications running on an application server.

The client and server communicate using the HTTP protocol to transfer HTML according to the request and response model. Each client request is of the type "/console/*.ppc".

The application server directs requests for any `.ppc` to the controlling servlet (called the ConsoleServlet) which in turn redirects the request to the `pocketpc.jsp` file. The `pocketpc.jsp` file in turn redirects the request to the JSP as specified in the uientity. The JSP in turn renders an HTML response, which is displayed by the Sterling Supply Chain Applications Mobile Client. For information on extending the mobile device screens, see Appendix D, "Mobile User Interface Extensibility Reference".

Figure 1–2, "Mobile Architecture" illustrates this architecture.

Figure 1–2   Mobile Architecture
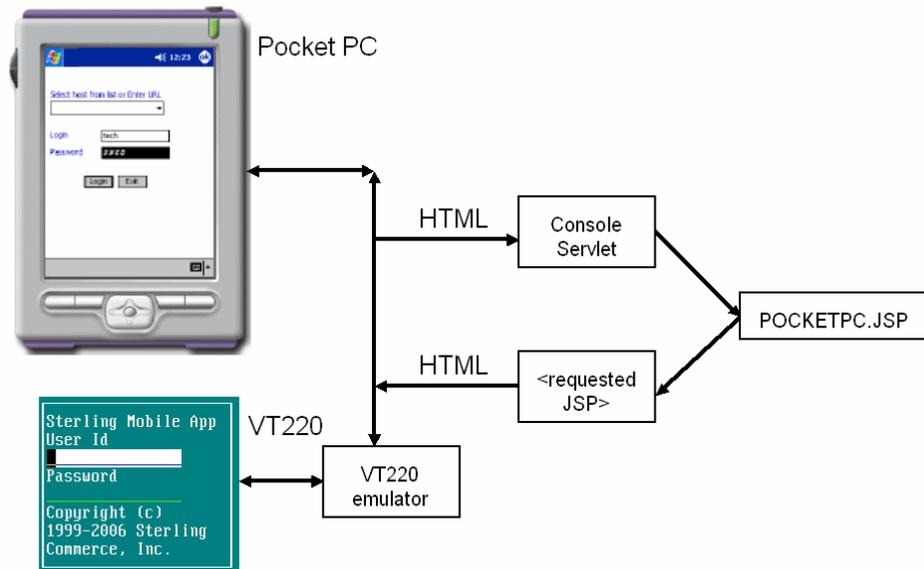
1.5 The Sterling Supply Chain Applications Database

Database extensibility enables you to add columns and tables to capture additional data.

For more information, see Chapter 7, "Extending the Sterling Supply Chain Applications Database".

1.6 Sterling Supply Chain Applications Transactions

Sterling Supply Chain Applications provide a mechanism for processing and resolving errors during data transformation and transportation. This mechanism, called the Service Definition Framework, enables access to the following transactional processes:

- System APIs exposed by Sterling Supply Chain Applications

- Event handlers to route the data that is published by Sterling Supply Chain Applications to the transport services layer

- Time-triggered transactions that monitor and execute tasks as needed

The Service Definition Framework provides error checking through the log4j utility, which writes both trace and debug information to a log file.

In addition, you can extend Sterling Supply Chain Applications by creating the following types of custom code:

- Extended (custom) APIs

- User exits that override default business algorithm

- User exits that extend the business algorithm used by APIs and time-triggered transactions

- Custom time-triggered transactions

2

Getting Started

Before customizing Sterling Supply Chain Applications, it is important to set up the environment in which you create your customizations. This chapter explains how to prepare your environment to perform customizations to the user interface and database.

2.1 Before You Begin

This guide assumes that you have already installed Sterling Supply Chain Applications. It also assumes that you are familiar with creating and running the `yantra.ear` file in deployment mode, as described in the *Sterling Supply Chain Applications Installation Guide*. It also assumes that you are familiar with the standard installation, otherwise known as the factory default installation. Throughout this guide <YANTRA_HOME> refers to the directory where you have installed Sterling Supply Chain Applications and <YFS_HOME> refers to the <YANTRA_HOME>/Runtime folder.

2.2 Understanding the Development Environment

The development environment you need depends on the type of work you are doing.

If you intend to customize Sterling Supply Chain Applications, you need a test environment that enables you to ensure that your changes work as you intend. To save development time, you can customize your test environment to run Sterling Supply Chain Applications in *development mode*.

Development mode saves time by enabling your application server to automatically load the latest version of edited JSP files directly from

specific directories rather than reading them from the `yantra.ear` file. This enables you to customize and test iteratively, without having to repeatedly create a `yantra.ear` file.

Development mode also enables you to immediately test UI customizations.

If you extend the database, include the `yfsdbextn.jar` file before the `yantrashared.jar` in the CLASSPATH in all scripts.

The way you set up the development environment depends on the application server you use.

> **Note:** The customer can run the application server against the /Runtime directory, but before creating the <YANTRA_HOME>/Runtime, make sure that all the configurations and extensions for the Foundation or PCAs are saved to the <YANTRA_HOME>/Applications folder. This will allow you to recreate the <YANTRA_ HOME>/Runtime without losing your configurations and extensions. For more information about <YANTRA_ HOME>/Runtime, refer to *Sterling Supply Chain Applications Installation Guide*.

2.3 Preparing a Development Environment on WebLogic

To enable WebLogic to run Sterling Supply Chain Applications without creating an EAR, you must define an application in WebLogic with the appropriate settings and then configure your startup script to setup the CLASSPATH required by Sterling Supply Chain Applications.

Setting up Sterling Supply Chain Applications directory structure enables WebLogic to read from files directly rather than from the `yantra.ear` file. The following steps provide the necessary details on setting up the scripts to run WebLogic in exploded mode.

> **Note:** All configuration or extensions should be completed in the respective <YANTRA_ HOME>/Applications/<Foundation_or_specific_PCA>/ folder. The Sterling Supply Chain Applications runtime should then be updated using one or both of the following ant commands:
>
> ```
> ant -f bin/buildRT.xml mergebin
> ```
>
> ```
> ant -f resources/buildRT.xml mergeresources
> ```
>
> The files should then be copied to and from the appropriate <YFS_HOME> (Sterling Supply Chain Applications runtime) folder.

To set up the scripts to run WebLogic in exploded mode:

1. Copy the `web.xml` file and the `weblogic.xml` file from the `<YFS_ HOME>/descriptors/weblogic/WAR/WEB-INF` directory to the `<YFS_ HOME>/webpages/WEB-INF` directory.

2. Copy the `ycpapibundle.properties` file and `ycpapibundle_<lang>_ <country>.properties` (if applicable) from the `<YFS_ HOME>/resources` directory to the `<YFS_ HOME>/webpages/yfscommon/` directory.

   a. If a PCA is installed, copy the `<PCA_Code>bundle.properties` file from the `<YFS_HOME>/resources` directory to the `<YFS_ HOME>/webpages/yfscommon` directory. For example, for Sterling COM PCA, copy the `ycdbundle.properties` file.

3. Copy all files from the `<YFS_HOME>/resources/extn/` directory to the `<YFS_HOME>/webpages/extn` directory.

4. If you want to use online help in the development mode, copy the `<YFS_HOME/documentation/online_help` folder to `<YFS_ HOME>/webpages/yfscommon` folder.

5. On Unix create a soft link in the `<YFS_HOME>/webpages/WEB-INF` directory to the `<YFS_HOME>/lib` directory as:

   ```
   ln -s /lib lib
   ```

6. On Windows, create a lib directory in the `<YFS_ HOME>/webpages/WEB-INF` directory and copy all the *.jar files from

the `<YFS_HOME>`/lib directory to the `<YFS_HOME>/webpages/WEB-INF/lib` directory.

Now you need to configure WebLogic as described below. If you need further information, see the WebLogic documentation.

WebLogic must be configured to enable the server to read from the `<YFS_HOME>/webpages/` directory. The necessary steps for configuring WebLogic to run Sterling Supply Chain Applications in exploded (non-ear) mode for your development environment are given below

> **Note:** Sterling Supply Chain Applications deployed in the exploded mode work the same as the applications deployed the EAR mode. There are no performance implications that are specific to exploded mode.
>
> Sterling Commerce recommends the EAR mode of deployment in production. In case an application server hosts multiple applications, there is no interference for jars or classes across applications. This is because each application is packaged or deployed as a single EAR file that is isolated from other application packages (EARs). However, in the exploded mode, the class that is first added to the class path is always considered.

To configure WebLogic to run Sterling Supply Chain Applications in the exploded mode:

1. Start your WebLogic server and open the WebLogic system console. This can be accessed using a URL similar to the following:

   ```
   http://<hostname or ip-address>:<port number of your WebLogic
   Server>/console
   ```

2. Log in to the console using the system administrator ID and password for your WebLogic server.

3. In the Change Center panel, click Lock & Edit.

4. In the Domain Structure panel, click Deployments.

If there are existing deployments of Sterling Supply Chain Applications, delete them. To delete the existing deployments, you must stop and delete the deployment.

   a. Stop the existing deployments of Sterling Supply Chain Applications.

      – Check the box of the applicable deployment you want to delete.

      – Click Stop and select Force Stop Now from the pop-up menu.

      – In Delete Application Assistant, click Yes.

      – In Messages, the message "Selected Deployments have been requested to stop" displays.

   b. Delete the existing deployments of Sterling Supply Chain Applications:

      – Check the box of the applicable deployment you want to delete.

      – Click Delete.

      – In Delete Application Assistant, click Yes.

      – In Messages, the message "Selected Deployments were deleted. Remember to click Activate Changes after you are finished" displays.

      – In the Change Center panel, click Activate Changes.

5. Click Install.

6. In Location:, browse to the `<YFS_HOME>` directory.

7. Choose the `webpages` folder and click Next.

8. Choose Install this deployment as an application and click Next.

9. In General, enter the name of the deployment.

10. In Security, choose DD Only: Use only roles and policies that are defined in the deployment descriptors.

11. In Source accessibility, Choose I will make the deployment accessible from the following location.

In Location:, make sure that the location points to the `<YFS_HOME>/webpages` directory.

12. If you want to deploy the Sterling Supply Chain Applications without reviewing the configurations:

- Click Finish.

- In the Change Center panel, click Activate Changes.

If you want to review the deployment configurations:

- Click Next.

- Choose Yes, take me to the deployment's configuration screen.

- Click Finish.

   In Messages, the message "Settings updated successfully" displays.

- Click Save. The Sterling Supply Chain Applications are deployed on the WebLogic server.

13. Restart the WebLogic server. You can access Sterling Supply Chain Applications.

2.4 Preparing a Development Environment on WebSphere

When using WebSphere, you can test the modifications that you have made to the Sterling Supply Chain Applications as described in this section. The necessary steps for configuring WebSphere to run Sterling Supply Chain Applications in exploded (non-ear) mode for your development environment are given below:

1. Deploy the yantra.ear, using the documentation provided by IBM. During deployment, WebSphere copies all the contents of WAR and EAR files to the `<WAS Home>/AppServer/profiles/<PROFILE_ NAME>/installedApps/` directory.

After deployment, any files copied to the `<WAS Home>/AppServer/profiles/<PROFILE_NAME>/installedApps/` directory can be modified as needed. For example, if you are extending a custom code written as part of the database extensibility, can be directly moved to the appropriate directory under `<WAS Home>/AppServer/profiles/<PROFILE_NAME>/installedApps/` directory for testing. IBM calls this ability to modify and move files as needed "hot deployment".

The custom JSPs that are written as part of UI extensibility can be directly incorporated into the `yantra.war` file.

2. If you want to use online help in the development mode, copy the `<YFS_HOME/documentation/online_help` folder to `<YFS_HOME>/webpages/yfscommon` folder.

3. Stop the application server before performing the following steps.

   a. Locate the file `yantra.jar` in the `<WAS Home>/AppServer/profiles/<PROFILE_NAME>/installedApps/<CELL_NAME>/<APP_NAME>` directory and rename the file.

   b. Create a directory named `yantra.jar` in the `<WAS Home>/AppServer/profiles/<PROFILE_NAME>/installedApps/<CELL_NAME>/<APP_NAME>` directory.

   c. Extract the jar files in `yantra.jar` file to `yantra.jar` directory using the command:

      ```
      jar -xvf yantra.jar
      ```

      This directory is specified as a root of the classpath.

   d. The custom templates and class files can be copied to this directory.

4. Copy your customized files (for example, custom transaction code, template extensions, localization literal files), to the appropriate `<WAS Home>/AppServer/profiles/<PROFILE_NAME>/installedApps/<CELL_NAME>/<APP_NAME>/yantra.jar/<Module_Name>` directory.

   For example, if you have some customizations in the Catalog module, add the files in `<WAS Home>/AppServer/profiles/<PROFILE_NAME>/installedApps/<CELL_NAME>/<APP_NAME>/yantra.jar/ycp` directory.

5. Re-package the `yantra.jar` with the following command:

   ```
   jar -cvf yantra.jar
   ```

6. Test your customizations using the following standards:

Table 2–1   WebSphere Hot Deployment Test Mode

If you modify...	In these files...	Then...
Startup parameters	properties	Restart WebSphere
UI extensibility	JSP, JavaScript, CSS, theme XML	Load dynamically
Localization literals	alertmessages and ycpapibundle properties files	Restart WebSphere
Database extensions	entity XMLs	You need to rebuild the yfsdbextn.jar file and include them in your CLASSPATH, then restart WebSphere
APIs and other template files	template XMLs	Load dynamically

2.5 Configuring the UI Cache Refresh Actions

In a standard deployment of Sterling Supply Chain Applications, any configuration changes made within the Resource Configurator do not take effect until the application server has been restarted. This means that testing your UI extensions can be a time-consuming activity. Therefore, Sterling Supply Chain Applications provide actions within the Resource Configurator that enable you to refresh the resources so that your modifications can be tested immediately.

These actions can only be enabled in a development environment. They will not work in a deployment environment. This section explains how to enable these actions. These actions should be disabled in a deployment environment.

To configure the resource cache refresh actions:

1. Open the `yfs.properties` file and set the `yfs.uidev.refreshResources` property to `Y`.

2. Update the Sterling Supply Chain Applications Runtime for these modifications. For more information about updating Sterling Supply Chain Applications Runtime, see the *Sterling Supply Chain Applications Installation Guide*.

3. This enables the following actions on the Resource Configurator Tree in the Configurator:

    Refresh Cache icon - Refreshes all resources.

    Refresh Entity icon - Refreshes the selected entity resource and its child resources.

    For instructions on using the Refresh Cache icons, see "Testing UI Customizations" on page 15.

2.6 Developing and Testing in the Development Environment

Now that you have set up your development environment, you are ready to begin customizing Sterling Supply Chain Applications, using the directions provided throughout this guide. This section explains how to use the icons that enable you immediately test UI customizations.

> **Note:** After any modifications are made to the following files, the application server must be restarted:
>
> - YFSDataType.xml file
> - yfsdatatypemap.xml file

2.6.1 Testing UI Customizations

After making changes to UI Resources within the Resource Configurator, you can test your changes immediately by using the Cache Refresh icons.

To use the cache refresh icons:

1. Make changes as needed.

2. Select the refresh cache icon that fits your needs as follows:

    — If you want to update one entity and its child resources - Select the specific entity and choose the *Refresh Entity Cache* icon

    — If you want to update all resources - Choose the *Refresh Cache* icon

3. Log into Sterling Supply Chain Applications again to test your changes.

2.7 Deploying Your Customizations

After you are satisfied with all of the customizations you have made to the UI and database, build the `yantra.ear` file on your development system and test it. Then deploy your changes to a production system and test them again. For information about deploying Sterling Supply Chain Applications, see the *Sterling Supply Chain Applications Installation Guide*.

Before deploying in production mode, ensure that the `yfs.properties` file has the correct settings. For example, ensure that the cache refresh icons specified in the `yfs.uidev.refreshResources` property is set to `N`.

2.8 Using Sterling Supply Chain Applications with Microsoft COM+

When using Sterling Supply Chain Applications with COM+, you need to create and configure Sterling Supply Chain Applications on your Windows server. You also need to create and install a client proxy.

2.8.1 Creating a Sterling Supply Chain Applications COM+ Application on Windows

To create a Sterling Supply Chain Applications COM+ application on a Windows server:

1. From the Windows Start menu, navigate to Administrative Tools > Component Services.

2. From the Component Services tree, navigate to Component Services > Computers > My Computer > COM+ Applications and then right-click COM+ Application. Select New > Application.

3. After the "Welcome to COM Application Install Wizard" appears, choose Next.

4. Choose Create an Empty Application.

5. In the Create Empty window enter the following and then choose next:

    -   For Application Name enter Sterling Supply Chain Applications.

- • For Activation Type choose Server Application. This ensures that the components are started as dedicated processes.

6. In the Set Application Identity window choose This User and enter the appropriate Windows user name and password. This user is the identity under which the application will run. Make sure that the user belongs to the Administrators group. Choose Next and then Finish.

   The newly created Sterling Supply Chain Applications COM+ application appears under the Component Services tree.

Now you can add components to your Sterling Supply Chain Applications COM+ application.

2.8.2 Adding Components to a COM+ Application

To add components to a Sterling Supply Chain Applications COM+ application:

1. From the Component Services tree, navigate to Component Services > Computers > My Computer > COM+ Applications > Sterling Supply Chain Applications > Components and then right-click Components. Choose New > Components.

2. After the "Welcome to COM Component Install Wizard" appears, choose Next.

3. Choose Install New Component.

4. Browse to `<YANTRA_ HOME>/Applications/Foundation/bin/YIFComApi.dll`. Select it and choose Open. Then choose Next and then Finish.

5. Make sure that your system path contains the directories that store the following DLLs:

   - • YIFJNIApi.dll

   - • msvcrt.dll

   - • msvcp60.dll

   - • jvm.dll (usually found under <JAVA_HOME>/jre/bin/hotspot on the client machine)

6. Make sure that the `<YANTRA_ HOME>/Applications/Foundation/resources` directory has the following properties files:

- yifclient.ini
- yifclient.properties
- yfs.properties

2.8.3 Configuring the Sterling Supply Chain Applications COM+ Service

To configure the Sterling Supply Chain Applications COM+ service:

1. From the Component Services tree, right-click the newly created Sterling Supply Chain Applications COM+ Application.

2. Choose Properties.

   The Sterling Supply Chain Applications Properties dialog box appears.

3. Choose the Advanced tab.

4. Under Server Process Shutdown panel, select Minutes Until Idle Shutdown and enter the time in minutes after which you want the process to shut down and then choose OK.

5. Double-click the Sterling Supply Chain Applications COM+ application.

6. Double-click Components.

7. Right-click YIFComApi.YIFComApi.1 and choose Properties.

   The YIFComApi.YIFComApi.1 Properties dialog box appears.

8. Choose the Activation tab.

9. Choose Enable object pooling.

10. In the Object pooling section, enter the Minimum and Maximum pool sizes based on the Component usage. Configure pooling to make optimal use of your hardware resources. The pool configuration can change as available hardware resources change.

11. Choose Enable Just In Time Activation.

    JIT activation activates an instance of an object just before the first call is made to it and then immediately deactivates the instance after it finishes processing its work.

2.8.4 Creating the Client Proxy

To create the client proxy:

1. Right-click your Sterling Supply Chain Applications COM+ application and choose Export.

2. After the "Welcome to COM Application Export Wizard" appears, choose Next.

3. In the Application Export window, enter the path and file name where the export `.MSI` file is to be created.

4. Choose Export as Application Proxy. Choose Next and then Finish.

2.8.5 Installing the Client Proxy

To install the client proxy:

1. Make sure the following DLL files are in your system path:

   • `<YFS_HOME>/bin/YIFJNIApi.dll`

   • `<YFS_HOME>/bin/msvcrt.dll`

   • `<YFS_HOME>/bin/msvcp60.dll`, `jvm.dll` (located under `<JAVA>`/jre/bin/hotspot on the client machine)

   • Double-click the `.MSI` file to install the component on the client.

   > **Note:** This method has been deprecated in Release 7.7. The recommended method to call APIs and Services from a Microsoft Windows COM+ environment is through webservices or over http.

3

Customizing the Console JSP Interface

The Sterling Presentation Framework enables you to change the way information is rendered (or displayed) in the Application Consoles, without changing the way it functions.

This chapter describes how to customize the Sterling Supply Chain Application Consoles user interface to suit your business needs. Each task is accomplished through a combination of configuration through the Sterling Supply Chain Applications Configurator user interface and editing HTML code in the JSP files of the screens you want to modify.

> **Note:** The HSDE (Execution Console Framework) screens are not extensible.

For information on using the Sterling Supply Chain Applications Configurator, see the *Sterling Supply Chain Applications Platform Configuration Guide*. For information on functions used within the JSP files, see Appendix C, "Console JSP Interface Extensibility Reference".

> **Important:** When customizing the interface, copy the standard resources of the Sterling Supply Chain Applications and then modify your copy. **Do not modify the standard resources of the Sterling Supply Chain Applications.**

The following applications are shipped with Sterling Supply Chain Applications:

- Sterling Supply Chain Application Consoles - the standard UI for creating, tracking, and viewing orders, item inventory, and returns

- Sterling Supply Chain Applications Configurator - the UI for configuring Sterling Supply Chain Applications. For additional configuration information, see Chapter 4, "Customizing the Configurator Swing Interface".

3.1 Before You Begin

Before you begin user interface extensibility, familiarize yourself with guidelines listed in this section in order to help ensure a successful experience and enable you to work more quickly and with fewer errors.

Compare JSPs

As part of Sterling Supply Chain Applications extensibility, you can extend JSP files of the current release.

When Sterling Commerce releases service packs or major releases and, in some cases, a user interface hot fix, you need to complete a JSP comparison and reconciliation. During the JSP comparison process you must compare the following files:

1. The original JSP shipped with your original Sterling Supply Chain Applications product version

2. The same JSP as Step 1 that you have extended for your environment

3. The same Sterling Supply Chain Applications JSP shipped as part of the new release

In order to facilitate the reconciliation process, Sterling Commerce recommends that you consider purchasing a difference tool that is capable of performing a 3-way comparison. For example, an inexpensive tool that can be purchased is Araxis. Please note that Sterling Commerce does not re-sell this difference tool nor do we warranty or guarantee this difference tool. It's merely provided as an example of an inexpensive tool and you are advised to do your own research on 3-way comparison tools and choose the one appropriate for your specific needs.

Prepare for Smooth Upgrades and Easy Maintainability

- Do not change the resource definitions of any of the resources shipped as part of the standard default configuration. Always make a

copy through the Sterling Supply Chain Applications Configurator and then change the copy.

- Do not change any of the JSP files, JavaScript files, and icon JAR files that are supplied with Sterling Supply Chain Applications. If you do, your changes may be lost during upgrades.

- When creating new views, consider issues regarding ease of maintenance as well as ease of creation. When you create a new view, inner panel, and so forth, it is possible to link to the JSPs supplied by Sterling Supply Chain Applications. But in future releases, Sterling Commerce may add more resources to these JSP, which means you must monitor software changes and update your configuration to account for these changes.

  For example, if you create a new inner panel and link it to the Order Detail Header JSP, (`/om/order/detail/order_detail_header.jsp`), this JSP file expects specific APIs to be called. Specifically, it expects the `getOrderDetails()` API to be called with certain fields in the output template and the `getScacAndServiceList()` API to be called for the Scac And Service drop-down field.

  If you use this inner panel, you must configure these two APIs with the necessary fields in the output template. Then, if Sterling Commerce adds fields to this JSP file in future releases, you may need to modify your configuration to account for these changes.

Build in Usability

- Any new views you develop should look and behave like the product views, so before you begin developing, gain an understanding of how the default views behave. For more information on the basic product look and feel, see "Introducing the Screen Layout and Behavior" on page 35.

Prepare Your Development Environment

In order to start the customization process, you must perform prepare the development environment to accommodate for your changes. See "Understanding the Development Environment" on page 7.

Understand How to Find Reference Materials

The Sterling Presentation Framework consists of several JSP, JavaScript, and class files. These files contain several functions declared as `public`.

However, only some of these functions are `exposed`. While performing user interface extensions, only the `exposed` functions should be used. See the following locations to determine whether a function is `exposed`:

- Java Classes - see the *Sterling Supply Chain Applications Javadocs*

- JavaScript - see "JavaScript Functions" on page 647

- JSP - see "JSP Functions" on page 598

- JSP Tags - see "JSP Tag Library" on page 635

3.2 Duplicate Request Handling and Page Tokens

In any Sterling Supply Chain Applications user interface that involves HTML forms, there is a possibility of duplicate requests to be unintentionally created by the browser. Problems can occur when requests that result in updating some data are unintentionally submitted multiple times. For example, a duplicate request may get generated when the browser is manually refreshed by the user. The user intended to perform the update only once, however, multiple requests for the same update can be generated.

To avoid any potential problems created by duplicate requests, the Sterling Supply Chain Applications infrastructure framework uses page tokens to ensure only the original request is processed. When a user logs into the Sterling Supply Chain Application Consoles, a unique identifier called page token is assigned to the screens that are opened. This page token is validated every time an update request is sent by the browser. If the token does not match the one assigned to the screens earlier, then the request is ignored. Once a successful update request has occurred, the page token is reassigned to a new identifier. Therefore, any further requests that contained the old token will be ignored. This safeguard is built into the Sterling Supply Chain Applications infrastructure framework so you do not need to worry about this when performing customizations to the Sterling Supply Chain Application Consoles.

3.3 Defining Centralized Themes

When a user successfully signs in, the standard Sterling Supply Chain Applications Home Page is opened. The look and feel of the Home Page is determined by the user who has signed in and the theme associated with their user ID. The theme determines the fonts and colors to use in

rendering graphical user interface elements such as screens, labels, and table headers.

Sterling Supply Chain Applications supply the following standard themes:

- Sapphire (default)
- Earth
- Jade

A theme is defined centrally through CSS and XML files, depending on which application uses it. This means you should not hard code colors and fonts of the user interface controls to individual screens. Instead, use the directions provided in this guide. Each theme is used throughout the Application Consoles and Configurator. Ideally when defining themes, you should create the same theme for both applications, in order to ensure a consistent user experience.

Table 3–1   Theme File Types

Application	Required Theme Files
Sterling Supply Chain Application Consoles	• CSS - for HTML pages • XML - for displaying the colors and fonts used in graphs, charts, and maps. • _exui.xml - for displaying customized themes in the execution UIs.
Sterling Supply Chain Applications Configurator	• XML - throughout

Do not modify Sterling Supply Chain Applications standard themes. Create your own CSS and XML files as described below. In order to determine which classes are used to define the controls for each style, see "Controls and Classes" on page 584.

If you require themes that are customized for different locales, make sure you create a new file for each of the locale codes you require. This is because for each specific theme, users can switch locales. For more information on defining and using locale codes, see "Customizing the Sign In Screen" on page 27.

To create a new theme:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/template/api/sapphire.xml` theme file to `<YANTRA_ HOME>/Applications/Foundation/template/api/`**extn**`/<theme>.xml` .

2. Edit your new XML file to define the values that you want to use for colors and fonts for each Color Name and Font Name element you want to display within the Sterling Supply Chain Applications Configurator and within any bar or pie charts.

   > **Note:** The fonts defined in the new theme XML must exist on the system you are using.

3. Copy the `<YANTRA_ HOME>/Applications/Foundation/webpages/css/sapphire.css` theme style sheet to `<YANTRA_ HOME>/Applications/Foundation/webpages/css/<theme>.css`.

4. Edit your new style file to specify the new colors and fonts added to in the new theme file.

5. Copy the `<YANTRA_ HOME>/Applications/Foundation/template/api/sapphire_ exui.xml` file to `<YANTRA_ HOME>/Applications/Foundation/template/api/`**extn**`/<theme>_ exui.xml`

6. Edit your new XML file to define the values that you want to use for colors and fonts for each Color Name and Font Name element you want to display within the Sterling Supply Chain Applications Configurator and within any bar or pie charts.

   > **Note:** Be aware that modifying fonts may impact other areas, requiring you to make further changes, such as to the size of fields or other UI elements that use that font. For guidance, see "CSS Theme Classes" on page 594.

7. Update the Sterling Supply Chain Applications Runtime for these modifications. For more information about updating Sterling Supply

Chain Applications Runtime, see the *Sterling Supply Chain Applications Installation Guide*.

8. From the Sterling Supply Chain Applications Configurator menu, configure the theme.

After creating the new theme, a user can choose it from a drop-down list of themes on the Sterling Supply Chain Application Consoles user interface. For information on how users can define their own themes, see the *Sterling Supply Chain Applications Platform User Guide*.

3.4 Customizing the Sign In Screen

The Sign In screen is the first page a user sees when they start Sterling Supply Chain Applications. The Sign In screen is made up of the following files:

- start.jsp - opens login.jsp in a new browser window and removes the browser tool bar.
- login.jsp - includes logindetails.jsp
- logindetails.jsp - is extensible

These Sign In screen files open in the manner shown in Figure 3–1:

Figure 3–1 Sign In Screen Logic
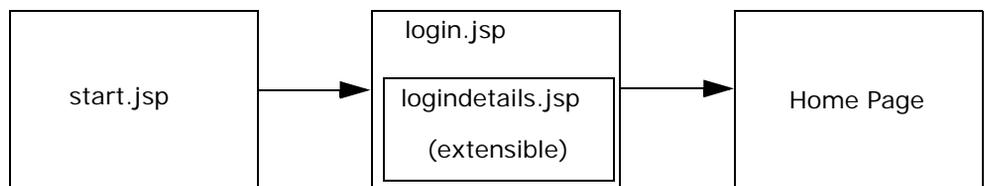
The primary purpose of the Sign In screen is to enable authorized users to log into Sterling Supply Chain Applications and establish their own personalized session. When the user signs in, the following properties are set up for their session:

- Locale settings
- Security access controls

- User properties such as preferred theme, organization, and so forth

3.4.1 Setting the Locale

The locale you set determines the language in which on-screen literals are displayed. The `start.jsp` file displays the Sign In screen in the default language. Then after a user logs in, the language displayed on-screen comes from the setting specified within their locale.

A multinational organization may require several languages for localization. For example, it may be the case that the standard installation is in German, and a user's locale is French. In that instance, the `start.jsp` and `login.jsp` files are displayed in German and the user's home page is displayed in French. The language displayed within the `login.jsp` file is determined by the resource bundle it uses.

The user's entire session is displayed in the language specified in his profile until user selects a different locale or the session is terminated. If the user selects a different locale, Sterling Supply Chain Applications dynamically changes any literals associated with the new locale.

The session terminates either when the user logs out or when a connection times out. When the session terminates, the Sign In screen displays the standard language.

3.4.2 Configuring Logins for a Specific Locale

If your user community is multilingual, it makes sense to display the login page in every possible language. From the login page, the user can choose which language they want to see displayed on the Sign In screen.

To display an internationalized sign in screen:

1. Create a corporate HTML page that contains hyperlinks that correspond to each language that you want to display.

2. For each hyperlink, insert an <a href> tag to link to the locale specific page, using the syntax below:

```
/yantra/console/start.jsp?LocaleCode=<locale_code>
```

The <locale_code> value must match the entry specified in the Locale Code field in the Configurator. For information about the

Locale Code field and its suggested syntax, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

For example, to open the Sign In screen in Canadian French, use the following syntax:

```
/yantra/console/start.jsp?LocaleCode=fr_CA
```

This shows the Sign In screen in the locale specified. After the login succeeds, the application opens in the user's default locale.

3.4.3 Configuring User Sign in from an External Application

When integrating Sterling Supply Chain Applications with external applications, you may need to enable a user to automatically log into Sterling Supply Chain Applications from an external application. You do this by configuring a link from your external application's portal that automatically connects to Sterling Supply Chain Applications.

When choosing how to integrate the two applications, consider the following scenarios:

* Logging in a user and opening their home page

* Logging in a user and opening a specific view

To integrate Sterling Supply Chain Applications with an external application:

Within the external application's portal, insert one of the following URLs that best suits your needs:

– If you want to launch Sterling Supply Chain Applications, log in as a Sterling Supply Chain Applications user, and open their home page, use the following syntax:

```
/yantra/console/start.jsp?UserId=<LoginID>&Password=<PassPhrase>
```

– If you want to launch Sterling Supply Chain Applications, log in as a Sterling Supply Chain Applications user, and open a specific view, use the following syntax (which illustrates opening the default order search view):

```
/yantra/console/start.jsp?UserId=<LoginID>&Password=<PassPhrase>
&Redirect=/console/order.search
```

In either case, if the login fails, the browser opens the Sterling Supply Chain Applications Sign In screen, which prompts the user to enter their Login ID and Password.

3.4.4 Supporting External Authentication

If users must log into Sterling Supply Chain Applications transparently using a domain password, Sterling Supply Chain Applications support third-party single sign-on applications. In the event your environment does not support a single sign-on application, Sterling Supply Chain Applications also support external authentication and internal authentication (by default). For information on using external authentication, see the *Sterling Supply Chain Applications Installation Guide*. For programming information on single sign-on, see the *Sterling Supply Chain Applications Javadocs*, and "Single Sign On" on page 554.

3.5 Adding Corporate Logos

The standard application conveys Sterling Supply Chain Applications branding logos throughout the application. You can change the branding logos displayed on screen in the following locations:

- Sign In screen

- Menu Bar

- About Box

Use a unique image in each instance, since they all use images of different sizes.

For the Application Consoles, you must specify the relative URL of the image including the path as it exists in the JAR file. For example, the default icons used in the console are located in the `yantraiconsbe.jar` file. The path inside the JAR file of many of the icons is `console/icons`. Therefore, the image specified in the Resource Configurator for a console resource is `/yantra/console/icons/consoleresource.gif`.

> **Note:** After making modifications to the required files, you must create or update the Sterling Supply Chain Applications Runtime. For more information about updating Sterling Supply Chain Applications Runtime, see the *Sterling Supply Chain Applications Installation Guide*. Before creating or updating the Sterling Supply Chain Applications Runtime, make sure that all the configurations and extensions for the Foundation or PCAs are completed in the Foundation or individual PCA folder in the <YANTRA_ HOME>/Applications directory. This enables you to recreate the Sterling Supply Chain Applications runtime without losing your configurations and extensions.

3.5.1 Customizing the Sign In Screen Logo

The Sign In screen displays a corporate logo, which you may want to customize.

To customize the logo on the Sign In screen:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/webpages/console/logindetails .jsp` file to `<YANTRA_ HOME>/Applications/Foundation/webpages/`**extn**`/console/loginde tails.jsp` file.

2. Open the new `logindetails.jsp` file and search for `YANTRA_ICON` to find the <img> tag referring to the Sterling Commerce logo. Change the <img> tag to point to your corporate logo, specifying the path as `/yantra/console/icons/` (for example, for `MyLogo.gif` you would use `/yantra/console/icons/MyLogo.gif`).

3. Copy your image file to the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/icons/ console/icons/` directory.

4. From the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/icons/` directory, archive the entire console directory into a `yantraiconsbe.jar` file so that the path of the file inside the jar is `/console/icons`.

5. Put the `yantraiconsbe.jar` file under the `<YANTRA_ HOME>/Applications/Foundation/webpages/ extn/icons/` directory.

3.5.2 Customizing the Menu Bar Logo

The Menu Bar displays on every screen that is not a pop-up dialog box.

When a user logs into Sterling Supply Chain Applications, the `getMenuHierarchyForUser()` API is called and the output is stored in the `session` object. This reduces any overhead for building the menu for each screen, and it enables the Menu Bar to be configurable at the user level, (for information on configuring the Menu, see "Customizing the Menu Structure" on page 82).

To customize the logo on the Menu Bar:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/webpages/common/menubar.jsp` file to `<YANTRA_ HOME>/Applications/Foundation/webpages/`**`extn`**`/common/menubar. jsp`.

2. Open the new `menubar.jsp` file and search for `YANTRA_LOGO` to find the <img> tag referring to the Sterling Commerce logo. Change the <img> tag to point to your corporate logo.

3. Copy your image file to the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/icons/ console/icons/` directory.

4. From the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/icons/` directory, archive the entire console directory into a `yantraiconsbe.jar` file.

3.5.3 Customizing the About Box Logo

The About Box indicates the version number of an application. Sterling Supply Chain Applications ship a standard About Box that a user can access by clicking the logo displayed on the Menu Bar shown in Figure 3–3. Figure 3–2, "Application Consoles About Box" shows the Application Consoles About Box.

Figure 3–2   Application Consoles About Box

When displaying the About Box, the application reads the version number based on a pre-determined logic. It is strongly advised that you retain the logic of displaying the version number when customizing the logo.

If you have purchased any Packaged Composite Applications (PCA) from Sterling Commerce, the version number of that product also appears in the About Box.

If you want to add corporate branding information to the About Box, you can customize the logo displayed.

To customize the logo on the About Box:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/webpages/console/about.jsp` file to `<YANTRA_ HOME>/Applications/Foundation/webpages/`**extn**`/console/about.jsp`.

2. Write your own HTML in your new `about.jsp` page.

3. Change your copy of the `menubar.jsp` file so that the `onclick` event of the `<img>` tag calls your new `about.jsp` file.

> **Tip:** In order for the About Box (and other pop-up windows) to maintain an appearance and behavior that is consistent with the rest of the application, use the `window.showModaldialog()` function for displaying pop-up windows.

3.6 Introducing the Screen Layout and Behavior

In the Sterling Supply Chain Application Consoles, a screen is made up of an *anchor page* which defines how a screen should display any inner panels it contains. Within Sterling Supply Chain Applications, anchor pages, inner panels, and the logic associated with them are referred to as *views*. Figure 3–3 depicts the standard Home Page, which shows the components used in a search view.

Figure 3–3   Standard Home Page View
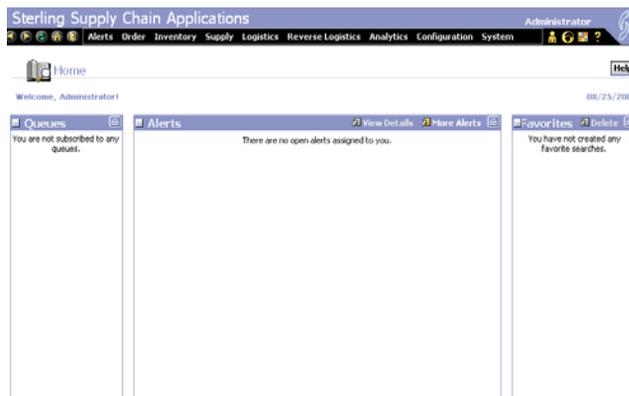
The following types of views are available in the Sterling Supply Chain Application Consoles:

• Search view

- List view

- Detail view

For detailed descriptions of these views, see "Introducing the Types of Screens" on page 38, which describes them in detail.

3.6.1 Screen Organization

All screens follow the same overall organizational pattern. In general, the Sterling Supply Chain Applications are laid out with a Menu Bar at the top and side-by-side search and list views below it. Figure 3–4 shows the typical organization of a screen.

Figure 3–4   Standard Screen Layout

Search View

List View
or Detail View
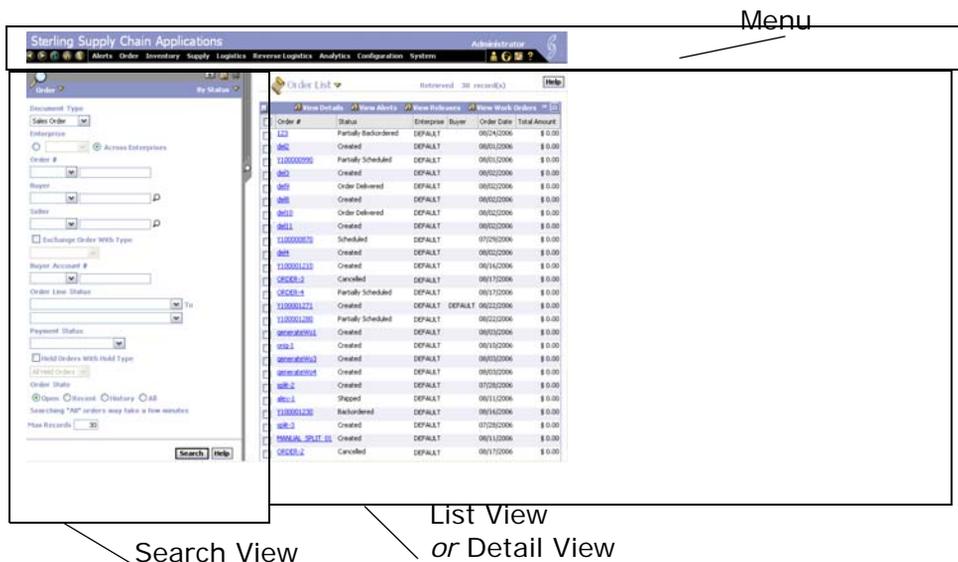
Menu

> **Note:**   Deviating from the standard organization of the screen layout, such as placing the menu on the left side of the screen, is not supported.

3.6.2 Screen Navigation Behavior

Sterling Supply Chain Applications screen navigation behavior follows a standard, consistent pattern. Figure 3–5 shows the flow of navigation logic from one inner panel to the next.

Figure 3–5   Screen Navigation Behavior

1. The search view fills the left side of the screen.

2. The list view fills the right side of the same screen and displays the search results. From there you can navigate to a Details Screen.

3. The detail view fills the entire screen, replacing both the Search and list views. Links or icons in the detail view invoke a pop-up window.

4. Closing the pop-up window returns the focus to previous view.

3.6.3 Changing the Screen Navigation Behavior

By default, functions are exposed by the Sterling Presentation Framework to provide hyperlinks. For a list of these functions, see Appendix C, "Console JSP Interface Extensibility Reference". Typically, the links are sufficiently configurable that you can change the view that the link points to without changing the code.

To learn how to modify the screen navigation behavior, see "Customizing Screen Navigation" on page 83.

The following sections describe the types of screens that are included in Sterling Supply Chain Applications.

3.7 Introducing the Types of Screens

The screens contained within an inner panel all have business and programmatic logic associated with them. This section first describes the business-related aspects of each view and then it describes the programming logic associated with each view.

3.7.1 Types of Extensible Business Entities

You can extend the screens of any type of *entity*. An entity is an associated group of UI displays and program logic that pertain to specific business practices. In general, each entity has a corresponding console. Sterling Supply Chain Applications contain the following high-level business entities for which you can create views:

- Alert Console

- Adjust Inventory Console

- Create Order Console

- Create Picklist Console

- Create Purchase Order Console

- Create Return Console

- Create Template Console

- Exception Console

- Inbound Shipment Console

- Inventory Console

- Manifest Console

- Order Console

- Outbound Shipment Console

- Plan Console

- Purchase Order Console

- Return Console

- System Management Console

- Template Console

These business-related entities can be broken down into more granular details. For example, Order Management can be broken down to more granular entities, like order, shipment, and container. You can also create your own entities.

The Resource Configurator enables you to customize and create entities that suit your business needs. Figure 3–6 shows how entities appear in the Resource Configurator. The hierarchical order is based on the default order of navigation.

Figure 3–6   Entities within the Resource Configurator
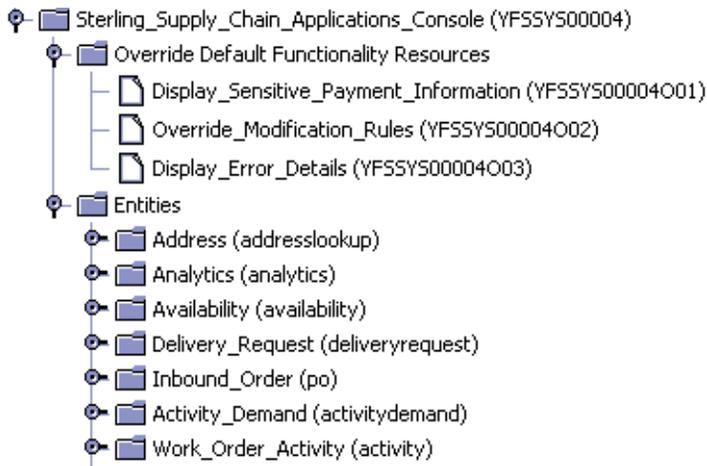
Each entity has a default search view, list view, and detail view. A default view is determined by the ordering of these views within the Resource Configurator.

For example, if the Order entity has four search views, the default search view is determined as the one with the lowest resource sequence number among the four search views.

Under each entity resource, you can configure one detail API and one list API. The detail API configured is automatically called when a detail view of that entity is opened. The list API is called when a list view of that entity is opened.

You can prevent this default API from being called for a specific view by selecting the Ignore Default API parameter in the resource configuration screen.

For more information about resource sequencing, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

> **Note:** Online help is not available for custom views. When a custom view is created, the Help button is displayed with a tool tip informing the users that help is not available. You can suppress the help for custom screens, by selecting the SuppressHelp option when configuring the resources for your screen.

3.7.2 Search Views

A search view contains a list of criteria from which to query. Figure 3–7 shows a standard search view.

Figure 3–7   Search View

The search view is made of one or more of the following elements:

- JSP pages - render the screen. For detailed information on JSPs, see Appendix C, "Console JSP Interface Extensibility Reference".

- APIs - populate drop-down menus. For each API, you can specify input parameters and an output template. For more information on configuring API resources used by the Application Consoles, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

3.7.3 List Views

The results of a search are displayed in a list view. There are two types of list views: regular list views and advanced list views.

Regular List View

Figure 3–8 shows a standard list view.

Figure 3–8   List View

A regular list view consists of a heading area (that permits view switching) and an Action Bar that contains the Actions and the list body. By default, the list body is set to display up to 30 records on the screen. If users want to see more results displayed on the screen, they can choose to display up to a maximum of 200 records. If you want to give all users the ability to display more than 200 records, you can set a new maximum number by editing the yfs.properties file as described within "Changing the Maximum Number of Records Displayed" on page 51.

A list view is made of one or more of the following elements:

- JSP pages - render the body. It is the only part of the view that doesn't belong to the Sterling Presentation Framework.

- Additional APIs - are called for that list view.

- Actions - See "Types of Actions" on page 44.

Within an entity, all subordinate list views use the same list API defined at the entity level. In addition to a list API at the entity level, you can define additional APIs for each list view. It is possible to configure a list view that does not call the default list API. In such a case, the template configured for the list view is not used (since the list API itself is not called).

Advanced List View

If additional information needs to be listed, you can create an advanced list view. To a user, an advanced list view looks similar to a regular list view. However, the advanced list view is actually defined as a detail view resource. This enables an advanced list view to have all of the same features as a standard detail view.

Figure 3–9 shows an advanced list view. To a user, an advanced list view looks and feels like a regular list view, but an advanced list view is actually a detail view, composed of multiple inner panels, a save or

update feature, and an ability to provide custom hyperlinks, but without the "Showing 1 of N" component.

Figure 3–9   Advanced List View
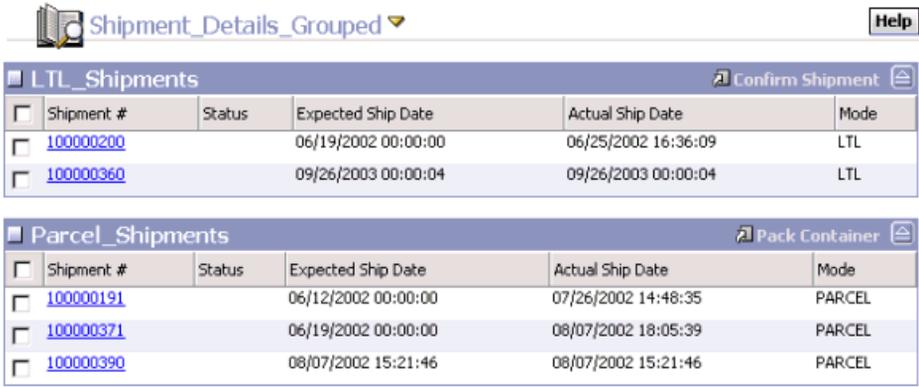
3.7.4 Detail Views

A detail view shows more specific information about an entity. Figure 3–10 shows a standard detail view.

Figure 3–10   Detail View
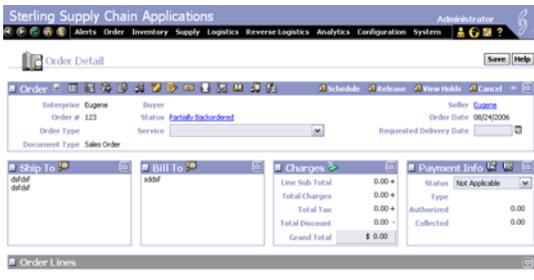
A detail view consists of one or more inner panels. An anchor page defines the layout of these inner panels. For example, the Order Detail

anchor page includes all inner panels relevant to Order Detail and defines how they should be laid out horizontally.

The anchor page is optional. If an anchor page is not specified, the inner panels within the detail view are automatically laid out one after another vertically.

Each inner panel consists of fields along with a title bar that contains zero or more words (known as "actions") that enable the opening of a new detail window, and zero or more icons (known as "views") that enable the opening of a pop-up window. This inner panel title bar with actions and views is described as the "Action bar".

A detail view makes use of multiple APIs. These APIs are considered as follows:

1. You can define a detail API for each entity.

2. Each detail view can specify multiple APIs to call.

3. Each detail view consists of inner panels and each inner panel can specify multiple APIs to call.

Additionally, you can configure a detail view so that it does not call the default detail API. For more information about configuring details see the *Sterling Supply Chain Applications Platform Configuration Guide*.

A detail view consists of one or more of the following elements:

- Inner panel - one or more.

- JSP page - anchor page defined by the view ID within the Resource Configurator.

- Save Action - one or more. See Section 3.7.5, "Types of Actions" on page 44.

3.7.5 Types of Actions

From the list views and the detail views, the following actions are possible:

- Go to another view - opens a view in a modal dialog

- Call a script - calls the specified JavaScript

- Call an API - calls the specified API

- Call an API in rollback-only mode - calls the specified API in a rollback-only mode. This action can be used in a "what if" kind of scenario. For example, a customer service representative can check the total price of an order by adding an additional line without committing the transaction in the database. This option can be enabled in the Resource Configurator. For more information on enabling this option refer to *Sterling Supply Chain Applications Platform Configuration Guide*.

You can use each action by itself but they are more useful combined together. For example, you can configure a JavaScript and an API for an action. In this scenario, the specified API is invoked only when the specified JavaScript returns `true`.

In another example, you can configure an API and a view for an action. In this scenario, the API is invoked and regardless of its success, the configured view is invoked. This view opens within the same browser window.

> **Note:** You can resize the pop-up browser window using the JavaScript call `window.dialogWidth`, `window.dialogHeight` and specifying the width and height as required.

If you want Sterling Supply Chain Applications to open a view only when an API returns success, you must configure that specifically as described in this section.

To configure a view to open only on the success of an API:

1. Configure an action to call an API.

> **Note:** You must define your action code with a certain naming convention to avoid any discrepancy with the Sterling Supply Chain Applications factory default action codes. For example, the custom action code can be prefixed with a `EXTN_`.

2. Configure your JSP to detect the specified attribute in the output XML of the API and store that attribute in a specified custom attribute of the HTML.

3. On the client side, write an `onLoad` function that searches for the specified custom attribute and then uses the `showDetailFor()` JavaScript function to switch to the screen you want.

3.8 Customizing Views

You can either create a new view on your own or use an existing view as a template. Using a template is far easier.

3.8.1 Creating an Entirely New View

You may also choose to create entirely new resources. This method is *only* recommended for advanced users since it does not enable you to take advantage of the structure and templates provided by existing resources.

> **Important:** You must *completely* understand the structure of resources before attempting to create new resources.

To create a new resource:

1. From the Sterling Supply Chain Applications Configurator menu choose Platform > Presentation > Resources. This displays the Resource Configurator.

2. Choose Create New.

3. Customize your new resource, keeping in mind the following rules:

    - The highest level of resource you can create is an *entity* resource.

    - Under your custom entity, it is possible to create a complete hierarchy of resources for it, which consists of views, inner panels, and so forth.

    - Generally, an entity resource should contain a corresponding list API and detail API.

    - A detail view should contain at least one inner panel.

    - An action resource must fit one of the following conditions:

        * have an API subresource

*   be configured with a view ID

*   be configured with a JavaScript to call

Alternatively, if you want an easier way to customize Sterling Supply Chain Applications, you can create new views, actions, or icons within entities available in the Sterling Supply Chain Application Consoles, using templates as described in the following sections.

3.8.2 Creating a View from a Template

Creating a view from a template involves copying and modifying an existing view. This is the easiest route, whether you just need a minor change to a view, or an entirely new view, as it provides the following benefits:

- Reduces the amount of work you must do

- Ensures a standard look and feel

- Ensures proper execution of application logic

> **Note:** When customizing the views you have to ensure that the relevant property name in the `yfs.properties` file is set to "Y" before proceeding to click the refresh icon. For more information, see Section 2.5, "Configuring the UI Cache Refresh Actions"

> **Note:** When creating new views, consider issues regarding ease of maintenance as well as ease of creation. For more information, see "Before You Begin" on page 22.

3.8.3 Customizing a Search View

Customizing a search view involves minimal changes.

To customize a search view:

1. Navigate to the screen that you want to change, so you can determine its view ID.

2. Hover your mouse pointer over the screen's view title. The view title's tool-tip indicates the view ID.

3. From the Resource Configurator, navigate to the view ID and copy it, including the sub-resources when prompted.

4. Copy the JSP file for the original view into the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/<PathOfOriginal JSP>/` directory.

> **Note**: Customizing this view may require other files to be copied into the `extn` folder. A list of these required files can be seen by right clicking on the entity and getting a JSP List

5. From the Resource Configurator, navigate to your new view. Enter the relative path to your JSP file in the JSP field. For example:

`/extn/om/order/search/order_search_bystatus.jsp`

6. If you want your new search view to be the default view, resequence the new view so that the sequence number is a lower number than that of the original view.

7. Edit your JSP file as needed. See "Customizing JSP Files" on page 55.

8. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide.*

9. Select the refresh cache icon that fits your needs as follows:

 – If you want to update one entity and its child resources - Select the specific entity and choose the *Refresh Entity Cache* icon

 – If you want to update all resources - Choose the *Refresh Cache* icon

10. Log into Sterling Supply Chain Applications again to test your changes.

3.8.4 Customizing a List View

You can customize regular list views or advanced list views.

3.8.4.1 **Customizing a Regular List View**

Customizing a regular list view involves minimal changes. You can modify the elements displayed or the maximum number of records displayed.

To customize a regular list view:

1. Navigate to the screen that you want to change, so you can find out its view ID.

2. Hover your mouse pointer over the screen's view title. The view title's tool-tip indicates the view ID.

3. From the Resource Configurator, navigate to the view ID and copy it, including its sub-resources when prompted.

4. Copy the JSP file for the original view into the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/<PathOfOriginal JSP>`/ directory.

> **Note**: Customizing this view may require other files to be copied into the `extn` folder. A list of these required files can be seen by right clicking on the entity and getting a JSP List

5. From the Resource Configurator, navigate to your new view. Enter the relative path to your JSP file in the JSP field. For example:

`/extn/om/order/list/order_list_concise.jsp`

6. If you want your new search view to be the default view, resequence the new view such that the sequence number is a lower number than that of the original view.

7. Edit your JSP file as needed. See "Customizing JSP Files" on page 55.

8. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

9. Select the refresh cache icon that fits your needs as follows:

   ‒ If you want to update one entity and its child resources - Select the specific entity and choose the 🌼 *Refresh Entity Cache* icon

      – If you want to update all resources - Choose the *Refresh Cache* icon

10. Log into Sterling Supply Chain Applications again to test your changes.

3.8.4.2 Customizing an Advanced List View

Customizing an advanced list view is similar to creating a custom detail view. The advanced list can be displayed for any specific search view. For search views that include an advanced list view, when a user chooses the search button, the advanced list view of the entity opens instead of the regular list view. For any other search views, the regular list view opens.

To customize an advanced list view:

1. From the Configurator, define a new search view with the Show Detail flag checked.

When the user executes a search, the default detail view of the entity is displayed (rather than the default list view for the typical search screen).

2. Edit your JSP file as needed. See "Customizing JSP Files" on page 55.

When using the following controls, make sure to include the corresponding JSP functions of the Sterling Supply Chain Applications when binding the input fields on the search view to an XML. This ensure that the input fields are available as input to the APIs called within the advanced list screen in the yfcSearchCriteria namespace.

> **Note:** The standard JSPs used for search views may include functions other than those listed. When customizing an advanced list view, ensure that the controls use the functions listed here.

Table 3–2   Controls

Control	Function
Textbox	getTextOptions
Select dropdown	getComboOptions
Radio button	getRadioOptions

Table 3–2   Controls

Control	Function
Check Box	getCheckBoxOptions
Hidden Inputs	getTextOptions

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4. Create a custom detail JSP page as described in "Customizing a Detail View" on page 52. (This will be used as the advanced list view.)

5. Define all additional APIs needed to display the advanced list view.

   The yfcSearchCriteria namespace contains all the data needed for additional APIs, and its xml:/SearchData/@MaxRecords attribute specifies the Maximum Records value for the advanced list view.

6. Determine how the advanced list view should open.

3.8.4.3 Changing the Maximum Number of Records Displayed

By default, Sterling Supply Chain Applications display a maximum of 30 records for the user. If the user wants to see more results displayed on the screen, they can choose to display as many as 200 records maximum.

If you want users to be able to display more than 200 records, you can set a new maximum number of records by editing the yfs.properties file. Once you set a new upper limit in the yfs.properties file, this system-level setting applies to all users.

The behavior of displaying 30 records by default cannot be modified.

To modify the maximum number of records displayed:

Open the <YANTRA_ HOME>/Applications/Foundation/resources/yfs.properties file and edit the MaxRecords value as shown below:

```
#-------------------------------------------------------------
This property sets the number of records displayed on a list
screen. Increase the application server JVM heap settings if
```

```
these parameters are increased. Change will affect search
limits for all users.
#-------------------------------------------------------------
yfs.ui.MaxRecords=200
```

> **Note:** Increasing the upper limit to beyond 200 impacts performance. It also requires increasing the application server JVM heap.

After editing the `yfs.properties` file, you must update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide.*

3.8.5 Customizing a Detail View

A detail view shows more specific breakdown of information. It consists of one or more inner panels within an anchor page. Each inner panel consists of fields along with zero or more actions and zero or more icons. A detail view can make use of multiple APIs.

To customize a detail view:

1. Navigate to the screen that you want to change, so you can determine its view ID.

2. Hover your mouse pointer over the screen's view title. The view title's tool-tip indicates the view ID.

3. From the Resource Configurator, navigate to the view ID and copy it, including its sub-resources when prompted.

4. Copy the JSP file for the original view into the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/<PathOfOriginal JSP>/` directory.

> **Note**: Customizing this view may require other files to be copied into the `extn` folder. A list of these required files can be seen by right clicking on the entity and getting a JSP List

5. If the original view uses an anchor page and you want to customize it, use the Resource Configurator to navigate to your new view. Enter the relative path of your JSP file into the JSP field.

6. Edit the anchor page JSP file as needed. See "Customizing JSP Files" on page 55.

> **Note:** Ensure that the anchor page contains the Resource IDs of all of the inner panels you want included. See the *Sterling Supply Chain Applications Platform Configuration Guide*.

7. For each inner panel you must customize, perform the following steps:

   a. Copy the JSP file for the original inner panel into the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/<PathOfOrigi nalJSP>/` directory.

   b. From the Resource Configurator, navigate to your new inner panel. Enter the relative path to your JSP file in the JSP field. For example:

     `/extn/om/order/detail/order_detail_header.jsp`

   c. Edit your inner panel JSP file as needed. See "Customizing JSP Files" on page 55.

8. If you want your new detail view to be the default view, resequence the new view so that the sequence number is a lower number than that of the original view.

9. In order to make all links point appropriately to your new view, follow the steps in "Incorporating Your View across the Application" on page 66.

10. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications

runtime, see the *Sterling Supply Chain Applications Installation Guide*.

11. Select the refresh cache icon that fits your needs as follows:

- **–** If you want to update one entity and its child resources - Select the specific entity and choose the 🌸 *Refresh Entity Cache* icon

- **–** If you want to update all resources - Choose the 🔁 *Refresh Cache* icon

12. Log into Sterling Supply Chain Applications again to test your changes.

3.8.5.1 Blocking the Reason Code Popup from Order Detail screen

The SAVE action for the changes you make in the Detail Screen of Sterling Supply Chain Applications console results in a popup asking for the Reason Code for the changes made. The Reason Code popup can be blocked from appearing only for a custom screen.

Note: The popup editing would be possible only if it is a custom screen. The default screens of Sterling Supply Chain Applications are not editable.

To Remove the appearance of popup screen invoked by the 'SAVE' action:

1. From the Sterling Supply Chain Applications Configurator menu choose Platform > Presentation > Resources. This displays the Resource Configurator.

2. Choose Entity > Detail View >Action that you want to edit. In this case, SAVE is chosen as the action to be modified.

3. Remove the Java Script method that invokes the Popup screen.

3.8.6 Customizing JSP Files

JSP files contain the syntax that enable your HTML pages to display dynamically. The types of JSPs files correspond to search views, list views, and detail views. The standards for each JSP file depend on the type of screen that uses it.

JSP Files for Search Views

A JSP file for a search view typically contains tags that create input fields which permit users to enter search criteria. A search JSP file usually includes the following types of HTML controls:

- Labels

- Input fields

- Combo boxes

- Radio buttons

- Checkboxes

JSP Files for List Views

A JSP file for a list view typically contains only an HTML table with column headers and data cells. Most list views also contain checkboxes for use with the actions that can be performed on the records in the list. An XML key is usually constructed and associated with the checkboxes on table.

JSP Files for Detail Views

A JSP file for a detail view are typically the most complicated, since detail view often require a wide variety of controls. Detail views usually contain the same sort of HTML controls as a search view. In addition, detail views may also contain the following controls:

- Text areas

- Tables

- Graphs

> **Reminder:** In order to maintain a consistent look and feel
> throughout the product, use the same stylesheets (CSS
> files) throughout all of the JSP files you customize.

3.8.7 Understanding the JSP Files Used by Sterling Supply Chain Applications

The JSP files provided by Sterling Supply Chain Applications cover every typical use case scenarios that you can encounter, so they are useful templates for developing your own JSP files. As you examine the JSP files, note that they are modular, which enables you to quickly customize a view by assembling units of code together. In addition to following the JSP file template, when you need more technical details, see Appendix C, "Console JSP Interface Extensibility Reference".

Example 3–1 shows some code from a typical order JSP file, which can be used to render an Order list view.

Example 3–1   JSP File Contents

```
<%@taglib prefix="yfc" uri="/WEB-INF/yfc.tld" %>
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%@include file="/console/jsp/currencyutils.jspf" %>
<%@ page import="com.yantra.yfs.ui.backend.*" %>
<%@ include file="/console/jsp/modificationutils.jspf" %>


<script language="javascript"
src="/yantra/console/scripts/modificationreason.js"></script>

<table class="table" editable="false" width="100%" cellspacing="0">
    <thead>
        <tr>
            <td sortable="no" class="checkboxheader">
                <input type="hidden" name="userHasOverridePermissions"
value='<%=userHasOverridePermissions()%>'/>
                <input type="hidden" name="xml:/Order/@Override" value="N"/>
                <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
            </td>
            <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Status</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
```

```
            <td class="tablecolumnheader"><yfc:i18n>Buyer</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Order_Date</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Total_Amount</yfc:i18n></td>
        </tr>
    </thead>
    <tbody>
        <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
            <tr>
                <yfc:makeXMLInput name="orderKey">
                    <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
                </yfc:makeXMLInput>
                <td class="checkboxcolumn">
                    <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
                </td>
                <td class="tablecolumn">
<a href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
<yfc:getXMLValue binding="xml:/Order/@OrderNo"/>
</a>
                </td>
                <td class="tablecolumn">
                    <% if (isVoid(getValue("Order", "xml:/Order/@Status"))) { %>
                        [<yfc:i18n>Draft</yfc:i18n>]
                    <% } else { %>
                        <yfc:getXMLValue binding="xml:/Order/@Status"/>
                    <% } %>
                    <% if (equals("Y", getValue("Order",
"xml:/Order/@HoldFlag"))) { %>
                        <img class="icon"
onmouseover="this.style.cursor='default'"
<%=getImageOptions(YFSUIBackendConsts.HELD_ORDER, "This_order_is_held")%>/>
                    <% } %>
                </td>
                <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/></td>
                <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@BuyerOrganizationCode"/></td>
                <td class="tablecolumn"
sortValue="<%=getDateValue("xml:/Order/@OrderDate")%>"><yfc:getXMLValue
binding="xml:/Order/@OrderDate"/></td>
                <td class="numerictablecolumn"
sortValue="<%=getNumericValue("xml:/Order/PriceInfo/@TotalAmount")%>">
                    <%=displayAmount(getValue("Order",
"xml:/Order/PriceInfo/@TotalAmount"), (YFCElement)
```

```
request.getAttribute("CurrencyList"), getValue("Order",
"xml:/Order/@RulesetKey"), getValue("Order",
"xml:/Order/PriceInfo/@Currency"))%>
                </td>
            </tr>
        </yfc:loopXML>
    </tbody>
</table>
```

This example shows *include* files at the top. In order to access most of the common public JSP functions in the Sterling Presentation Framework, most JSP files only require a reference to the `<YANTRA_HOME>/Applications/Foundation/webpages/yfsjspcommon/yfsutil.jspf` file, as in the following example of an include statement:

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
```

To access the `yfsGet*Options()` JSP functions, you must reference the `<YANTRA_HOME>/Applications/Foundation/console/jsp/modificationutils.jspf` file, as in the following example:

```
<%@ include file="/console/jsp/modificationutils.jspf" %>
```

For more information on these functions, see "JSP Functions" on page 598.

For JavaScript functions, all public functions are automatically available to your JSP file.

3.8.7.1 UI View Across Document Type

Any console screen that is an entry point to a particular set of console screens (like the order search or order create screens) may need to incorporate the common JSP used for implementing the user interface view across document type feature. This feature allows existing console screens to be used when viewing information for different document types. This common JSP can be used when the entry point screens in your entity's console require any of the following fields:

• Document Type

• Enterprise Code

• Node (only for WMS screens where node a primary important field)

3.8.7.1.1 Using the common_fields.jsp

The `common_fields.jsp` (located in the `webpages/yfscommon` directory) provides many different features for displaying the commonly required fields. The `common_fields.jsp` should be included in the top of your JSP. JSP parameters should be passed to indicate what features you need for the particular usage of the JSP. For example, if you want to show the node field, then you pass the "ShowNode" parameter as "true". Table 3–3 indicates all of the valid parameters that can be passed to the `common_ fields.jsp`

Table 3–3   Valid Parameters for Common_fields.jsp

Parameter	Description
ShowDocumentType	Indicates whether the document type field should be displayed or not. Default: true
ShowEnterpriseCode	Indicates whether the enterprise code field should be displayed or not. Default: true
ShowNode	Indicates whether the node field should be displayed or not. Default: false
DocumentTypeBinding	Indicates the binding that should be set on the document type field. Default: xml:/Order/@DocumentType
EnterpriseCodeBinding	Indicates the binding that should be set on the enterprise code field. Default value: Xml:/Order/@EnterpriseCode.
NodeBinding	Indicates the binding that should be set on the node field. Default value: xml:/Order/@ShipNode.
RefreshOnDocumentType	Indicates whether the entire screen should refresh when a document type is selected. (See the "Screen Refreshing" section in this document for more information.) Default: false.
RefreshOnEnterpriseCode	Indicates whether the entire screen should refresh when an enterprise is selected. (See the "Screen Refreshing" section in this document for more information.) Default: false.
RefreshOnNode	Indicates whether the entire screen should refresh when a node is selected. (See the "Screen Refreshing" section in this document for more information.) Default: false.
ScreenType	Indicates the type of screen in which this JSP is being included. This information is used to set the appropriate classes and column layout of the fields inside the JSP. Valid values are "search" and "detail". Default: search.

Table 3–3   Valid Parameters for Common_fields.jsp

Parameter	Description
ColumnLayout	Indicates the number of columns that will be used to display the fields. The only valid values allowed are 1 and 3. Default: If ScreenType is "search", then the default column layout is 1. If ScreenType is "detail", then the default column layout is 3.
NodeLabel	Indicates what the screen label for the node field should be. Only valid when "ShowNode" is passed as "true". Default: Node.
EnterpriseListForNodeField	Indicates if the values that display in the enterprise code field should be based on the selection within the node field. This is used for WMS screens where the enterprise list must be a list of organizations that participate in the node's organization. The "RefreshOnNode" parameter should be passed as "true" when this parameter is "true" to ensure that the enterprise list will refresh when a node is selected. Only valid when "ShowNode" and "ShowEnterpriseCode" are "true". Default: false

> **Note:**   Any API called for fetching data for fields **within** the `common_fields.jsp` is done by the common JSP itself. There is no need to define resources in your screens for these APIs. For example, if you are showing the enterprise code field using the common JSP, there is no need to define the `getOrganizationList()` API within your screen's resources.

3.8.7.1.2   Screen Refreshing

For fields that depend on document type, enterprise code, or node, use the common JSP's screen refreshing features. For example, in the order search by status screen, there is a "From Status" field that displays a list of statuses for which you can search by. This list of valid statuses is different for different document types. Therefore, when the user selects a particular document type, a different list of statuses must appear in the field. Achieving this requires the following steps:

1. Include the common_fields.jsp at the top of the JSP. Pass the "RefreshOnDocumentType" parameter as "true" as follows:

```
<jsp:include page="/yfsjspcommon/common_fields.jsp" flush="true">
        <jsp:param name="DocumentTypeBinding"
value="xml:/OrderRelease/Order/@DocumentType"/>
```

```
        <jsp:param name="EnterpriseCodeBinding"
value="xml:/OrderRelease/Order/@EnterpriseCode"/>
        <jsp:param name="RefreshOnDocumentType" value="true"/>
    </jsp:include>
```

2. In the application XML for your entity, define the `getStatusList()` API under the view. Define the API so that it is not called by the infrastructure layer when the view is displayed (set the "LoopAPI" attribute to "Y"). Specify a dynamic attribute for the DocumentType attribute so that it will pass the document type selected in the field on the screen as follows:

```
DocumentType="xml:CommonFields:/CommonFields/@DocumentType"
```

3. In the JSP of your screen, call the `getStatusList()` API using the `callAPI` tag lib immediately after the common_fields.jsp is included.

> **Note:** You can refresh the entire screen using any of the common fields (document type, enterprise code, and/or node). There is a corresponding dynamic binding that must be specified for each
> (`xml:CommonFields:/CommonFields/@DocumentType`,
> `xml:CommonFields:/CommonFields/@EnterpriseCode`,
> `xml:CommonFields:/CommonFields/@Node`) respectively.

3.8.7.1.3 Other Common Field Features/Notes

- When a particular field is displayed using the common_fields.jsp, the appearance of the field depends on the number of records that the field needs to display. If there is only one record to display in the field, then the field will appear as a protected input that cannot be modified by the user. If there are 2 to 20 records to display in the field, then the field will appear as a combo box. If there are more then 21 records to display, then the fields will appear as a protected text box with a lookup icon next to it. In this case, the only way to change the value of the field is through the lookup. The reason for this behavior is so that the "screen refreshing" feature can work even when there are too many records to show in a combo box.

- If a user logged into the console is a node user, the node field will appear as a protect input that cannot be modified. Otherwise, the

node field will display all of the current logged in organizations owned nodes.

Screen Example

The following example shows how this common JSP would be used within a search screen where two fields on the search screen need to be refreshed whenever the enterprise code field changes.

In the JSP of all entry point screens, the common_fields.jsp will be included:

Example 3–2   Usage of JSP within a Search Screen

```
<table class="view">

    <jsp:include page="/yfsjspcommon/common_fields.jsp" flush="true">
        <jsp:param name="DocumentTypeBinding"
value="xml:/OrderRelease/Order/@DocumentType"/>
        <jsp:param name="EnterpriseCodeBinding"
value="xml:/OrderRelease/Order/@EnterpriseCode"/>
        <jsp:param name="ShowNode" value="true"/>
        <jsp:param name="NodeBinding" value="xml:/OrderRelease/Order/@Node"/>
        <jsp:param name="RefreshOnNode" value="true"/>
        <jsp:param name="RefreshOnEnterprise" value="true"/>
        <jsp:param name="EnterpriseListForNodeField" value="true"/>
    </jsp:include>
    <% // Now call the APIs that are dependent on the common fields (Doc Type,
Enterprise Code, and Node)
       // Product Classes and Unit of Measures are refreshed.
    %>
    <yfc:callAPI apiID="AP2"/>
    <yfc:callAPI apiID="AP3"/>

<tr>
    <td class="searchlabel"><yfc:i18n>field1</yfc:i18n></td>
</tr>
<tr>
    <td nowrap="true" class="searchcriteriacell">
        <select class="combobox" name="xml:/OrderRelease/@Field1QryType">
            <yfc:loopOptions
binding="xml:/QueryTypeList/StringQueryTypes/@QueryType" name="QueryTypeDesc"
value="QueryType" selected="xml:/OrderRelease/@Field1QryType "/>
        </select>
        <input type="text" class="unprotectedinput"
```

```
<%=getTextOptions("xml:/OrderRelease/@Field1")%> />
    </td>
</tr>
```

APIs are defined in the application XML:

```
<View ViewGroupID="YOMSXXX" Priority="3" Name="By_Item" ID="YOMSXXX"
JSP="/om/order/search/wms_by_item.jsp" OutputNode="Order">
      <APIList>
          <API Name="getQueryTypeList" OutputNode="QueryTypeList">
             <Input>
                <QueryType/>
             </Input>
             <Template>
                <QueryTypeList>
                   <StringQueryTypes>
                      <QueryType QueryType="" QueryTypeDesc=""/>
                   </StringQueryTypes>
                </QueryTypeList>
             </Template>
          </API>
          <API Name="getCommonCodeList" OutputNode="ProductClassList"
LoopAPI="Y">
             <Input>
                <CommonCode CodeType="PRODUCT_CLASS"
CallingOrganizationCode="xml:CommonFields:/CommonFields/@EnterpriseCode"/>
             </Input>
             <Template>
                <CommonCodeList>
                   <CommonCode CodeValue="" CodeShortDescription=""/>
                </CommonCodeList>
             </Template>
          </API>
          <API Name="getCommonCodeList" OutputNode="UnitOfMeasureList"
LoopAPI="Y">
             <Input>
                <CommonCode CodeType="UNIT_OF_MEASURE"
CallingOrganizationCode="xml:CommonFields:/CommonFields/@EnterpriseCode"/>
             </Input>
             <Template>
                <CommonCodeList>
                   <CommonCode CodeValue="" CodeShortDescription=""/>
                </CommonCodeList>
             </Template>
          </API>
```

```
            </APIList>
        </View>
```

3.8.8 Creating Inner Panels for a Detail View

Each inner panel comes from a separate JSP file and the anchor page JSP includes these files through the jsp:include JSP tag. If you configure more than one inner panel for a detail view, and if they must be simply laid out one below the other, the Sterling Presentation Framework provides a default anchor page to do that. In such a case, you do not have configure any JSP for the detail view.

Example 3–3 shows the typical syntax for including inner panels in an anchor page.

Example 3–3   Including Inner Panels in an Anchor Page

```
<table class="anchor" cellpadding="7px" cellSpacing="0">
<tr>
    <td colspan="2" >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I01"/>
        </jsp:include>
    </td>
</tr>
<tr>
    <td height="100%" width="75%">
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I02"/>
        </jsp:include>
    </td>
    <td height="100%" width="25%" addressip="true" >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true">
            <jsp:param name="CurrentInnerPanelID" value="I03"/>
            <jsp:param name="Path" value="xml:/OrderRelease/PersonInfoShipTo"/>
            <jsp:param name="DataXML" value="OrderRelease"/>
            <jsp:param name="AllowedModValue"
value='<%=getModificationAllowedValue("ShipToAddress",
"xml:/OrderRelease/AllowedModifications")%>'/>
        </jsp:include>
    </td>
</tr>
<tr>
    <td colspan="2" >
```

```
      <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
           <jsp:param name="CurrentInnerPanelID" value="I04"/>
      </jsp:include>
  </td>
</tr>
</table>
```

The innerpanel.jsp file provided by Sterling Supply Chain Applications contains the title bar that needs to be displayed for each inner panel, and stores the icons and action buttons available in the title bar of each inner panel.

JSP:Param JSP Tag Parameters

You can specify the following parameters to innerpanel.jsp file through the jsp:param JSP tag.

CurrentInnerPanelID - Suffix of the inner panel resource ID over the resource ID of the detail view. For example, if the detail view's resource ID is YOMD010, the inner panel's resource ID is YOMD010I01. In this example, you pass the I01 suffix to this JSP tag.

Title - Replaces the description of the inner panel resource ID configured.

IPHeight - Fixes the height of the inner panel. If the data grows beyond this height, a scroll bar automatically displays.

IPWidth - Fixes the width of the inner panel. If the data grows beyond this width, a scroll bar automatically displays.

Other than these attributes, the parameters that you specify here are automatically available to the JSP that is configured against the resource ID of the inner panel being included.

To create an inner panel:

1. Customize the view to which you want to add the inner panel. For details on customizing a detail view, see Section 3.8.5, "Customizing a Detail View" on page 52

2. Edit the anchor page JSP file to include the resource ID suffixes of any other inner panels you want to include.

The syntax of inner panel resource IDs <view ID's resource ID><Suffix, for example I01>.

You only need to refer to the suffix. The Sterling Presentation Framework forms the complete inner panel resource ID and includes the appropriate JSP.

3. From the Sterling Supply Chain Applications Configurator, create the inner panel resources under the newly created view.

4. From the Sterling Supply Chain Applications Configurator, create the actions and links as necessary.

5. Create the inner panel's JSP file, using a standard detail inner panel JSP as a template.

3.8.9 Incorporating Your View across the Application

When you are customizing an existing view, and want your customized view to replace the existing view across the application, you can do so without modifying the links, actions and icons that point to the existing view.

The existing links, actions and icons point to a view group ID. However, they do not point to a specific view when a screen is navigated to, using these links, actions and icons. The screen is opened as the view that has the lowest sequence number of all views with that view group ID.

Therefore, to replace an existing view across the application, ensure that your customized view has the same group ID as the original view and it has a sequence number that is lower than the original view sequence number.

For example, if you are replacing the standard order detail view with a customized view, the customized view must have a view group ID of `YOMD010`.

If your custom view is an additional screen that is not replacing an existing view, you must add your own links, actions or icons in the appropriate places such that it can be used to navigate to your screen.

Your view should have a unique view group ID. The links, actions or icons you create should point to this view group ID.

3.9 Customizing the Home Page

The Home Page is the default detail view of the Home entity. The standard Home Page has a menu view across the top, and three

side-by-side list views (User specific Queues, Alert and Favorite Searches) below it. Figure 3–3 shows the standard home page.

To customize the Home Page:

Follow either set of steps for customizing a detail view for the Home entity:

- "Customizing a Detail View" on page 52

- "Creating Inner Panels for a Detail View" on page 64

3.10 Creating a Custom Business Entity

You can create a custom entity by copying a standard entity and its sub-resources through the Resource Configurator.

> **Note:** Sterling Supply Chain Applications do not support resources with the same view group ID across entities. Hence, if you are copying entire entities for extending the screens, you must modify the view group IDs for all views, actions, links and icons for the new entities. This is to make sure that they do not conflict with the original entities you copied.

For example, if you want to create a custom business entity called Planned Order, you can use the following procedure:

To create a custom business entity:

1. From the Resource Configurator, navigate to an entity that you want to use a template. Choose an entity whose resources and sub-resources closely resemble the ones that you want to create.

2. Choose Save As to create the new set of resources for an entity, including sub-resources.

   When you save the entity, give it a unique prefix that will not conflict with any resource IDs that might ship with future releases. Sterling Commerce recommends that you choose any prefix except for one that begins with the letter *Y* (which is reserved for use by Sterling Supply Chain Applications).

3. Modify the description of the new entity resource. Modify the descriptions of any sub-resources. Note that the resource descriptions appear in the console; therefore, they can be localized as needed. To ensure that literals can be localized, use the resource description entered here as the resource bundle key in all of the appropriate resource bundles.

   Create entries for these newly created resource keys in the `<YANTRA_ HOME>/Applications/Foundation/resources/extn/extnbundle.pro perties` file and in the corresponding properties files for each locale.

4. Repeat Step 1 through Step 3 for the Related Entities of the entity being copied.

5. Update the Related Entities Resources under the newly created entity resources to point to the newly created resources.

6. In order to make all links point appropriately to your new view, follow the steps in Section 3.8.9, "Incorporating Your View across the Application" on page 66.

7. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

3.11 Using Extended Database Columns

You can customize views to incorporate any new columns added to the database. When extending the database, follow the directions in Chapter 7, "Extending the Sterling Supply Chain Applications Database".

If you want to add a field to the user interface, follow the directions for the view you want to change:

- Search view - Section 3.8.3, "Customizing a Search View".

- List view - "Customizing a List View" on page 48. After following the database extensibility rules, add the field to the output template configured for the API through the Resource Configurator.

- Detail view - "Customizing a Detail View" on page 52. After following the database extensibility rules, add the field to the output template configured for the API through the Resource Configurator.

3.12 Using the Override Entity Key Attribute

Often, optimal screen layout dictates the use an editable list of records in which the user can modify the items in the list. This table format usually maps to a list of XML elements in the API that handles the update for the screen. For example, the editable list of order lines on an Order Detail screen maps to the list of OrderLine elements accepted by the `getOrderDetails()` API.

By default, any action that appears on a detail view uses the current entity key as input to any API that is called for the action. For example, the Hold action on the Order Detail screen by default passes the current order header key to the `changeOrder()` API. There is a means to override which entity key is used for a specific action using the Override Entity Key attribute on an action resource. You can construct an input (usually a hidden input or a checkbox) on the JSP, give the input a value of an entity key constructed using the `makeXMLInput` JSP tag, and specify the name of that input as the Entity Key Name of the action. When that action is executed by the user, the new key is passed instead of the current entity key.

This feature is useful when a detail view shows the *details* of one entity and also contains an inner panel that displays a *list* of records for another entity. For example, the Order Detail screen shows details for the Order entity and also has an inner panel showing a list of order lines (which is a separate entity). Any action that appears on the order lines inner panel should not pass the order header key to the API. It should pass the order line key of the selected order lines.

For example, the following code might appear in an inner panel listing order lines for an order:

Example 3–4   Inner Panel Listing of Order Lines

```
<yfc:makeXMLInput name="orderLineKey">
<yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderLine/@OrderLineKey"/>
<yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey"/>
</yfc:makeXMLInput>
<td class="checkboxcolumn" >
<input type="checkbox" value='<%=getParameter("orderLineKey")%>'
name="chkEntityKey" />
</td>
```

This code creates a new entity key for the order line and associates this key to a checkbox named chkEntityKey. This name can then be specified in the action definition for any action appearing on the order lines inner panel, for example, the View Details action.

Note that this attribute frequently is used in conjunction with the Selection Key Name attribute that also can be defined for an action.

3.13 Posting Data for Editable Lists

APIs that take a list of elements in this way have different behavior based on the functionality that is required from the API. This different behavior must be handled by the user interface.

One way that an API may handle a list of elements is to completely replace the entire list each time the API is called. This means that the user interface must pass all attributes of each item whenever the API is called. This is accomplished by using the `IgnoreChangeNames()` JavaScript function. Calling this function when a JSP loads ensures that each single input on the screen is posted.

For example, edit your JSP file to contain the following code:

```
<script language="Javascript" >
IgnoreChangeNames();
</script>
```

In some cases an API might expect that a specific record be passed to the API only when some attribute has changed. Since the Sterling Presentation Framework does not automatically post any value that has not been changed by the user, it is quite possible that the XML constructed by the Sterling Service Definition Framework may contain an element with only the key attributes of a specific record. This can happen because the key attributes are usually hidden input objects in the JSP that are place within each row of the html table. Since they are hidden inputs, they are always posted to the API. Therefore, if the user does not change any of the attributes of a specific record in one row, only the key attributes are passed to the API. Some APIs consider this to be invalid input.

A Sterling Presentation Framework JavaScript function can be used to ensure that records for which no change has been made are not posted

to the API. The yfcSpecialChangeNames() function should be called when the page is unloaded.

For example, the following JSP code achieves this:

```
<script language=jscript>
window.document.body.attachEvent("onunload", processSaveRecordsForNotes)
</script>
```

The JavaScript function used in this example is defined as:

```
function processSaveRecordsForNotes() {
    yfcSpecialChangeNames("Notes", true);
}
```

In this example, the ID of the HTML table in the corresponding JSP is set to the literal Notes. The second parameter, true must be passed only if the ID Notes consists of a new blank row. The parameter should be set to false if you want to modify an existing row.

3.14 Retaining Unsaved Data in an Editable List

Some screens contain editable tables into which the user can enter more than one row of data. This data is typically saved once the user presses the save button and the save API is called. By default, if the save API throws an exception, the screen will refresh and the data that was entered in the editable table will not be retained. This section explains how you can retain the unsaved data even when an API exception occurs.

The following example explains how this feature can be implemented into an order instruction screen. The screen consists of two JSP files that have been customized to enable this functionality:

- order_detail_instructions_anchor.jsp

- order_detail_instructions.jsp

order_detail_instructions_anchor.jsp

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%@include file="/console/jsp/order.jspf" %>

<% setHistoryFlags((YFCElement) request.getAttribute("Order")); %>
```

```
<table class="anchor" cellpadding="7px"  cellSpacing="0" >
<tr>
    <td >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I02"/>
            <jsp:param name="getRequestDOM" value="Y"/>
        </jsp:include>
    </td>
</tr>
<tr>
    <td >
        <jsp:include page="/yfc/innerpanel.jsp" flush="true" >
            <jsp:param name="CurrentInnerPanelID" value="I01"/>
            <jsp:param name="ChildLoopXMLName" value="Instruction"/>
            <jsp:param name="ChildLoopXMLKeyName" value="InstructionDetailKey"/>
        </jsp:include>
    </td>
</tr>
```

A

Notice that a JSP parameter called getRequestDOM is passed as "Y" to the header JSP (in this case, order_detail_header.jsp). Any unsaved rows are saved in the request object as an XML file. By passing this parameter, the unsaved rows will be retrieved from the request object and merged with the API output that is used to load the screen initially. The refreshed screen will now contain the newly modified values that the user entered.

B

For the merge logic to work properly, the following parameters have to be passed to identify the elements and attribute on which to perform the merge. Please refer to Table 3–4.

Table 3–4   Parameters to be Passed

Parameter Name	Default Value	Comments
RootNodeName	Order	The root node of the output XML that will have the newly entered calues merged into it.
ChildLoopXMLName	OrderLine	The XML element that represents the repeating element name in the editable list.
ChildLoopXMLKeyName	OrderLineKey	The key that uniquely identifies the repeating XML elements. The merging logic uses this key to determine if the data for the specified element has been modified by the user.

order_detail_instructions.jsp

The highlighted code indicates the changes that will ensure that the data is retained after an API exception:

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%@ include file="/console/jsp/modificationutils.jspf" %>
<%@ include file="/yfsjspcommon/editable_util_lines.jspf" %>       C
<%@ page import="com.yantra.yfs.ui.backend.*" %>

<script language="javascript" src="../console/scripts/om.js"></script>

<% boolean isHistory=equals(resolveValue("xml:/Order/@isHistory"),"Y"); %>
<%
    boolean bAppendOldValue = false;
    if(!isVoid(errors) || equals(sOperation,"Y") || equals(sOperation,"DELETE"))
        bAppendOldValue = true;
%>
<table class="table" width="100%" cellspacing="0" <%if
(isModificationAllowed("xml:/@AddInstruction","xml:/Order/AllowedModifications")
) {%> initialRows="3" <%}%>>
<thead>
    <tr>
        <td class="checkboxheader" sortable="no">
            <%if( !isHistory ) { /* Then checkboxes are useless, since the
only action has been removed */ %>
```

D

```
                        <input type="hidden" id="userOperation" name="userOperation"
value="" />
                        <input type="hidden" id="numRowsToAdd" name="numRowsToAdd"
value="" />
                        <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
                <%}else { %>
                         
                <%}%>
            </td>
                <td class="tablecolumnheader" sortable="no"><yfc:i18n>Instruction_
Type</yfc:i18n></td>
            <td class="tablecolumnheader"
sortable="no"><yfc:i18n>Text</yfc:i18n></td>
        </tr>
</thead>
<tbody>
    <yfc:loopXML binding="xml:/Order/Instructions/@Instruction"
id="Instruction">
<%
    if(bAppendOldValue) {
        String sInstructionKey =
resolveValue("xml:Instruction:/Instruction/@InstructionDetailKey");
        if(oMap.containsKey(sInstructionKey))
    request.setAttribute("OrigAPIInstruction",(YFCElement)oMap.get
(sInstructionKey));
        } else
    request.setAttribute("OrigAPIInstruction",(YFCElement)pageContext.
getAttribute("Instruction"));
%>
<%
    if(!isVoid(resolveValue("xml:/Instruction/@InstructionDetailKey"))){ %>
        <tr>
        <yfc:makeXMLInput name="InstructionKey">
            <yfc:makeXMLKey binding="xml:/Instruction/@InstructionDetailKey"
value="xml:/Instruction/@InstructionDetailKey" />
            <yfc:makeXMLKey binding="xml:/Instruction/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
        </yfc:makeXMLInput>
        <td class="checkboxcolumn">
            <%if( !isHistory ) {%>
                <input type="checkbox"
value='<%=getParameter("InstructionKey")%>' name="chkEntityKey"/>
            <%}else { %>
                 
```

E

F

```
                <%}%>
            </td>
                <td class="tablecolumn">
                    <% String instTargetBinding =
"xml:/Order/Instructions/Instruction_" + InstructionCounter +
"/@InstructionType"; %>
                    <select <%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/@InstructionTyp
e")%>"  <%}%>
<%=yfsGetComboOptions(instTargetBinding, "xml:/Instruction/@InstructionType",
"xml:/Order/AllowedModifications")%>>
                            <yfc:loopOptions
binding="xml:InstructionTypeList:/CommonCodeList/@CommonCode"
name="CodeShortDescription" value="CodeValue"
selected="xml:/Instruction/@InstructionType" isLocalized="Y"
targetBinding="<%=instTargetBinding%>"/>
                    </select>
                </td>
                <td class="tablecolumn">
                    <table class="view" cellspacing="0" cellpadding="0">
                        <tr>
                            <td>
                                <textarea rows="3" cols="100"
<%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/@InstructionTex
t")%>"  <%}%>
<%=yfsGetTextAreaOptions("xml:/Order/Instructions/Instruction_" +
InstructionCounter + "/@InstructionText","xml:/Instruction/@InstructionText",
"xml:/Order/AllowedModifications")%>><yfc:getXMLValue
binding="xml:/Instruction/@InstructionText"/></textarea>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                <img align="absmiddle"
<%=getImageOptions(YFSUIBackendConsts.INSTRUCTION_URL, "Instruction_URL")%>/>
                                <input type="text" <%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/@InstructionURL
")%>"  <%}%>
<%=yfsGetTextOptions("xml:/Order/Instructions/Instruction_" + InstructionCounter
+ "/@InstructionURL",
"xml:/Instruction/@InstructionURL","xml:/Order/AllowedModifications")%>/>
                                <input type="button" class="button"
value="GO" onclick="javascript:goToURL('xml:/Order/Instructions/Instruction_
<%=InstructionCounter%>/@InstructionURL');"/>
```

```
                                    </td>

                                    <td>
                                          <input type="hidden"
<%=getTextOptions("xml:/Order/Instructions/Instruction_" + InstructionCounter +
"/@InstructionDetailKey", "xml:/Instruction/@InstructionDetailKey")%>/>
                                    </td>
                              </tr>
                              <tr>
                                    <td> </td>
                                    <td> </td>
                                    <td> </td>
                                    <td> </td>
                              </tr>
                        </table>
                  </td>
            </tr>
      <%    } else { %>
            <tr DeleteRowIndex="<%=InstructionCounter%>">
                  <td class="checkboxcolumn">
                        <img class="icon"
onclick="setDeleteOperationForRow(this,'xml:/Order/Instructions/Instruction')"
<%=getImageOptions(YFSUIBackendConsts.DELETE_ICON, "Remove_Row")%>/>
                  </td>
                  <td class="tablecolumn">
                        <input type="hidden" OldValue=""
<%=getTextOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@Action", "xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@Action", "CREATE")%> />
                        <input type="hidden"
<%=getTextOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@DeleteRow",  "")%> />
                              <select OldValue=""
<%=yfsGetTemplateRowOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@InstructionType", "xml:/Instruction/@InstructionType",
"xml:/Order/AllowedModifications", "ADD_INSTRUCTION", "combo")%>>
                                    <yfc:loopOptions
binding="xml:InstructionTypeList:/CommonCodeList/@CommonCode"
name="CodeShortDescription" value="CodeValue" isLocalized="Y"
selected="xml:/Instruction/@InstructionType"/>
                              </select>
                  </td>
                  <td class="tablecolumn">
                        <table class="view" cellspacing="0" cellpadding="0">
                              <td>
```

H

```
                                    <textarea rows="3" cols="100" OldValue=""
<%=yfsGetTemplateRowOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@InstructionText", "xml:/Instruction/@InstructionText",
"xml:/Order/AllowedModifications", "ADD_INSTRUCTION", "textarea")%>></textarea>
                            </td>
                            <tr>
                                <td>
                                        <img align="absmiddle"
<%=getImageOptions(YFSUIBackendConsts.INSTRUCTION_URL, "Instruction_URL")%>/>
                                        <input type="text" OldValue=""
<%=yfsGetTemplateRowOptions("xml:/Order/Instructions/Instruction_
"+InstructionCounter+"/@InstructionURL", "xml:/Instruction/@InstructionURL",
"xml:/Order/AllowedModifications", "ADD_INSTRUCTION", "text")%>/>
                                </td>
                            </tr>
                            <tr>
                                <td> </td>
                                <td> </td>
                                <td> </td>
                                <td> </td>
                            </tr>
                        </table>
                </td>
        </tr>
    <%   } %>
    </yfc:loopXML>
</tbody>
<tfoot>
    <%if
(isModificationAllowed("xml:/@AddInstruction","xml:/Order/AllowedModifications")
) {%>
    <tr>
        <td nowrap="true" colspan="3">
                <jsp:include page="/common/editabletbl.jsp" flush="true">
                    <jsp:param name="ReloadOnAddLine" value="Y"/>
                </jsp:include>
        </td>
    </tr>
    <%}%>
</tfoot>
</table>
```

I →

C The /webpages/yfsjspcommon/editable_util_lines.jspf file has been included in the list innerpanel jsp. This included file performs the merging of the user modified values and the original API output.

D JSP code has been added to ensure that the deletion of a row is handled properly.

E Hidden inputs have been added for user operation and number of rows to add.

F A map of the unique keys of the original repeating XML elements has been added has been added.

G The OldValue attribute of each editable field is set to the value of the original API output using the map previously created. This will ensure the value modified by the user will be posted to the save API.

H A new row is created in the editable table for each repeating XML element that does not contain a unique key attribute. This indicates that the row has not yet been saved in the database and was entered by the user before the API exception occurred.

I To add a rows dynamically when the user clicks on the plus icon under the table, the /common/editabletbl.jsp file is included next. Pass the ReloadOnaddLine attributes as "Y" to this JSP. Each time new rows added, the screen will be refreshed. Before these code changes, new rows would be added dynamically without refreshing the screen.

3.15 Adding a Lookup

Lookups enable users to choose from an assortment of options rather than typing in data. Figure 3–11 shows the available lookup icons and fields associated with them.

Figure 3–11   Lookup Icon
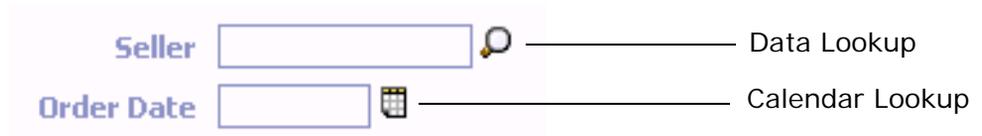
The Sterling Presentation Framework supports the following types of lookups:

- Single Field Lookup - Enables a user to search an entity for a specific value, select that value, and insert it into the appropriate single input field. Use the callLookup() JavaScript function. See "callLookup" on page 648.

- Calendar Lookup - Enables a user to select a date from a pop-up calendar. Use the invokeCalendar() JavaScript function. See "invokeCalendar" on page 660.

If you need a multiple field lookup, you can use the yfcShowSearchPopup() JavaScript function. This example shows product class, item ID, and unit of measure to be populated from an item lookup using the yfcShowSearchPopup() JavaScript function.

```
//this function should be called from "onclick" event of the icon next to
//item id field.
function callItemLookup(sItemID,sProductClass,sUOM,entityname)
{
    var oItemID = document.all(sItemID);
    var oProductClass = document.all(sProductClass);
    var oUOM = document.all(sUOM);
    showItemLookupPopup(oItemID, oProductClass, oUOM, entityname);
}

function showItemLookupPopup(itemIDInput, pcInput, uomInput, entityname)
{
    var oObj = new Object();
    oObj.field1 = itemIDInput;
    oObj.field2 = pcInput;
    oObj.field3 = uomInput;
    yfcShowSearchPopup('','itemlookup',900,550,oObj,entityname);
}
```

And in the lookup list view, call the following function to populate the field from which the popup was invoked:

```
function setItemLookupValue(sItemID,sProductClass,sUOM)
{
    var Obj = window.dialogArguments
    if(Obj != null)
    {
        Obj.field1.value = sItemID;
        Obj.field2.value = sProductClass;
        Obj.field3.value = sUOM;
    }
    window.close();
}
```

Use Lookup icons only with modifiable fields. When you incorporate a Lookup on field, place the appropriate icon directly to the right of the Lookup.

3.16 Creating a User-Sortable Table

In any table, a user can click a column heading to sort the results by that column. Clicking the same column heading again results in sorting the column in reverse order.

Table sorting does not create a new call to the API, but simply results in sorting of data already being displayed by that column.

To create a user-sortable table:
1. Use `table.htc` in the style attribute for the table. If you are using the default CSS files of Sterling Supply Chain Applications, you can use `class="table"` for the `<table>` tag.
2. The table should have `<tbody>` and `<thead>` tags that include `<td>` tags. If if you specify `sortable="no"` for any `<td>` tag in the `<thead>` tag, the column is not sortable.
3. For Date and Number, provide a separate `sortValue="<nonlocalized_value>"` in the actual `<tbody>` tag so that the data sorts properly.

Example 3–5 shows the tags needed for creating a user-sortable table.

Example 3–5   User-Sortable Table

```
<table class="table" editable="false" width="100%" cellspacing="0">
<thead>
<tr>
<td sortable="no" class="checkboxheader">
<input type="hidden" name="userHasOverridePermissions"
value='<%=userHasOverridePermissions()%>'/>
<input type="hidden" name="xml:/Order/@Override" value="N"/>
<input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
            </td>
            <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Buyer</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Order_Date</yfc:i18n></td>
        </tr>
    </thead>
    <tbody>
        <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
            <tr>
                <yfc:makeXMLInput name="orderKey">
                    <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
 value="xml:/Order/@OrderHeaderKey" />
                </yfc:makeXMLInput>
                <td class="checkboxcolumn">
                    <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
                </td>
                <td class="tablecolumn"><a
href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
              <yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
                </td>
                <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/></td>
                <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@BuyerOrganizationCode"/></td>
                <td class="tablecolumn"
sortValue="<%=getDateValue("xml:/Order/@OrderDate")%>"><yfc:getXMLValue
binding="xml:/Order/@OrderDate"/></td>
            </tr>
        </yfc:loopXML>
</tbody>
</table>
```

3.17 Adding Graphs and Pie Charts

Graphs and pie charts enable users to view a graphical representation of data. Graphs and charts derive their look and feel (appearance of the display colors and fonts) from the theme. For information about the theme and detailed instructions on how to modify one, see "Defining Centralized Themes" on page 24.

For displaying graphs of data, the Sterling Presentation Framework uses the jbchartx.jar file from Visual JChart. Since their tool may change in the future version, you should perform your own evaluation of which charting tool to use. The Sterling Presentation Framework does not provide a chart rendering API.

3.18 Customizing the Menu Structure

When creating customized screens, ensure that users can access them, either through a menu structure or through navigation.

Customizing the menu requires first laying out the structure of the menu through the Sterling Supply Chain Applications Configurator graphical user interface, and then specifying the literals in the resource bundle. A resource bundle is a file that contains all of the on-screen literals and messages. Sterling Supply Chain Applications provide the following standard resource bundles:

- ycpapibundle.properties - contains the literals used by the standard menu and on-screen messages. It cannot be modified.

- extnbundle.properties - contains the literals used by the custom menus. It can be customized and localized as needed.

You can create custom resource bundles that contain the modifications you make to the custom menu.

> **Note:** When customizing the menu, ensure that the menu description does not contain spaces or any other character that cannot be a valid resource bundle key as defined by java standards.

To create a custom menu:

1. Create a new menu using the graphical user interface described in the *Sterling Supply Chain Applications Platform Configuration Guide*.

2. Open the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/extnbundle.prop erties` file and add a key=Description mapping that uses an equal sign (=) to join the Description added in Step 1 to a key. For example, Special_Tasks=Special Tasks. The application reads the key in order to determine what to display on the screen.

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

3.18.1 Localizing the Menu Structure

You can customize the resource bundles to enable a single installation to support multiple languages. When localizing the Sterling Supply Chain Application Consoles, prepend `MENU_` to the menu keys you add to the resource bundle. For more information on localization, see the *Sterling Supply Chain Applications Localization Guide*.

> **Note:** When localizing the menu, ensure that the menu description does not contain spaces or any other character that cannot be a valid resource bundle key as defined by java standards.

3.19 Customizing Screen Navigation

You can customize how users navigate from entity to entity by configuring link or action resources. Links and actions can point to any detail view of any entity. Often the entity that you are navigating to requires a different entity key.

For example, the Order Detail screen has a list of order lines. The users can navigate to the order line detail screen by clicking the Order Line # hyperlink, or by selecting an order line and clicking the View Details action on the order lines inner panel. Since the order line detail screen requires a different entity key than the Order Detail screen, it is necessary to create another entity key in the order lines JSP to be used in the order line detail screen. Example 3–6 shows sample code for specifying screen navigation behavior.

Example 3–6   Screen Navigation

```
<yfc:makeXMLInput name="orderLineKey">
    <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderLine/@OrderLineKey"/>
    <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey"/>
    </yfc:makeXMLInput>
```

The way this entity key is passed to the order line detail screen differs for the hyperlink and action routes. For the hyperlink, this key is passed to the getDetailHrefOptions() function in the following manner:

```
<a <%=getDetailHrefOptions("L03", getParameter("orderLineKey"), "")%>>
   <yfc:getXMLValue binding="xml:/OrderLine/@PrimeLineNo"/></a>
```

Sometimes, you might want a detail view to behave differently based on some input parameter. For example, you might want to hide or show a field based on which is invoking the detail view.

You can call the getDetailHrefOptions() JSP function, using extra parameters that are passed to the target detail view. See "getDetailHrefOptions (with additional parameter)" on page 606 The format these extra parameters should be passed is name-value pairs separated by an ampersand ("&"). This is the standard format for passing parameters in a URL.

For the View Details action, the Selection Key Name must be set to the name of the checkbox that is created in the JSP. For example, in the JSP, the checkbox can be created as follows:

```
<input type="checkbox" value='<%=getParameter("orderLineKey")%>'
name="chkEntityKey"/>
```

The name of the checkbox is chkEntityKey. When configuring the action in the Resource Configurator, the Selection Key Name should be set to this to ensure the correct key is passed to the order line detail screen.

3.19.1 Disabling Direct Navigation to Detail Screens

From the Order, Purchase Order, or Return search screens provided by Sterling Supply Chain Applications, when the search results in only one record, the user is directed directly to the default detail view of the single record returned. This reduces the number of clicks required to get to the details of a particular record.

For example, the Order Search provided by Sterling Supply Chain Applications navigates directly to the Order Detail Screen if you enter a unique order number and press Search.

This type of navigation is controlled by the view definitions in the Resource Configurator. Only certain user interface entities support this type of navigation. Therefore, in the Resource Details of an entity resource, there is a "Support Direct List To Detail Navigation with One Record Returned" checkbox that indicates if a particular entity supports this feature. If an entity does support this feature, then the list view defined for that entity can turn this navigation on or off by enabling the "Support Direct List To Detail Navigation with One Record Returned" checkbox.

To disable this feature, create a copy of the existing list view, uncheck the checkbox, and ensure that the resource sequence is lower than the existing list view.

3.20 Developing Custom Event Handlers

You can create and plug in custom client-side validations to user interface controls for the following events:

- Events raised by Internet Explorer for that control or for that screen. For example, `onblur` event for input or `onunload` event for the page.

- Events raised by the Sterling Service Definition Framework for the page, for example, before saving the data in a detail view or before invoking search in a search view.

3.20.1 Control-Level Event Handler

Each text box has a behavior class associated with it that dynamically attaches a validation method to the `onblur` event (lost focus).

Each XML attribute must be tied to a data type and the infrastructure determines the data type based upon the XML attribute to which a data element is bound. The data type is used for data validations. For example, numeric fields should only accept number entry.

It is recommended that you limit client-side field-level validations to a minimum level. You can directly use `onblur="myValFun();"` in your HTML pages to perform custom validations. However, there is no guarantee that your function is called before the Sterling Presentation Framework function. Therefore, if you are using a numeric or date field, your function may return invalid data. You must call the Sterling Presentation Framework JavaScript utility functions to *first* validate the date and number before you proceed with your validations.

To create field-level validations:
1. Customize the view. See "Customizing a Search View" on page 47 or "Customizing a Detail View" on page 52.
2. Modify the customized JSP to include an event handler for the `onblur` event of the corresponding control.
3. Place the body of the JavaScript function in a separate JS file and include the JS file in your JSP.
4. Select the refresh cache icon that fits your needs as follows:
   – If you want to update one entity and its child resources - Select the specific entity and choose the 🐾 *Refresh Entity Cache* icon
   – If you want to update all resources - Choose the 🔁 *Refresh Cache* icon
5. Log into Sterling Supply Chain Applications again to test your changes.

3.20.2 Screen-Level Event Handler

The Sterling Presentation Framework uses the `onload` event on document.body. All other events are available for your use. The Sterling Presentation Framework uses the `attachEvent()` function to dynamically

attach event handlers to other events (for example, `onblur` on input). Therefore, in your JSP code, you can use other functions. If you want to invoke your own `onload` function, you can still use the `attachEvent()` function inside a script tag in your JSP as follows:

```
<script language=jscript src="/yantra/extn/scripts/om.js">
</script>
<script language=jscript>
window.document.body.attachEvent("onload", myFunc)
</script>
```

This causes the `myFunc()` function to execute when the HTML is loaded. Note that the body of the `myFunc` function must exist within the `<YANTRA_HOME>/Applications/Foundation/extn/scripts/om.js` file.

The Sterling Presentation Framework calls the Save action for a detail view when the Save icon is clicked. If you want to plug in your own custom event handler for this event, configure the action to call your JavaScript function. The function needs to be present in the `<YANTRA_HOME>/Applications/Foundation/webpages/extn/scripts/extn.js` file.

The Sterling Presentation Framework invokes the list view when the Search icon is clicked in a search view. If you want to plug in your own custom event handler for this event, attach your JavaScript function to the `onclick` event of the object returned by the `yfcGetSearchHandle()` JavaScript API when the page is loaded. The example below shows how to do this:

```
<script language=jscript src="/yantra/extn/scripts/om.js">
</script>
<script language=jscript>
//Get the handle to search button.
var oObj = yfcGetSearchHandle();

//The setParentKey function is defined inside om.js.
var sVal = oObj.attachEvent("onclick",setParentKey);
</script>
```

To create screen-level validations:

1. Customize the view in which you want to plug in validations.

2. Modify the customized JSP to include the `attachEvent`

3. Select the refresh cache icon that fits your needs as follows:

- If you want to update one entity and its child resources - Select the specific entity and choose the 🐾 *Refresh Entity Cache* icon

- If you want to update all resources - Choose the 🔄 *Refresh Cache* icon

4. Log into Sterling Supply Chain Applications again to test your changes.

3.21 Working with Document Types

The default Order console uses the Order document type (0001). When you create a new document type, you must also create a new entity with views and so forth. For a resource of Resource Type Entity, you can specify the document type.

Literals within the I18N tag resolve in a specific order. First, the key is prefixed with the document type and is looked up in the resource bundle. If no match is found, key is looked up as is, or without a prefix.

For example, if there is a I18N literal called Order_# and the current document type is Order (0001), Sterling Supply Chain Applications first try to resolve the resource key from the resource bundle for any entry for 0001_Order_#, and if not found, Order_#. This scheme enables you to reuse a specific JSP while still being able to change the literals that appear on the screen if they are specific to your document type.

To create a new set of screens for a new document type:

1. From the Resource Configurator, navigate to the Order entity (resource ID `order`).

2. Choose Save As to create the new set of resources from the Order entity, including its sub-resources.

   When you save the entity, give it a unique prefix that will not conflict with any resource IDs that might ship with future releases. Sterling Commerce recommends that you choose any prefix except for one that begins with the letter *Y* (which is reserved for use by Sterling Supply Chain Applications).

   Note that when copying resources for a new document type, you must copy *all* entities (including sub-resources) for the existing document type.

a. Determine which resources to copy, using the following SQL script:

```
select resource_id from yfs_resource
where resource_type='ENTITY' and document_type='0001'
```

b. Change the view group IDs for any Icons, Actions, Links and JavaScript functions that call the views. The view group ID should be changed to the new entity prefix and the view group ID that already exists.

3. Modify the description of the new entity resource to the description of your new document type. For all the sub-resources also, make appropriate modifications to the descriptions and create entries for these newly created resource keys in the `<YANTRA_ HOME>/Applications/Foundation/resources/extn/extnbundle.pro perties` file, and in the corresponding properties files for each locale.

4. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

5. Modify the Document Type for the new resources of resource type entity to the new Document Type.

6. Update the Related Entities Resources under the newly created entity resources to point to the newly created resources.

7. In order to make all links point appropriately to your new view, follow the steps in "Incorporating Your View across the Application" on page 66.

8. In Alert Console, if an alert is raised for an order, when you are viewing the alert details, you can click on the order and view the details. If you want that to link to also point to the main detail view of your document type, you must customize the Alert Details view.

a. Create a new Link ID under the new detail view of Alerts.

b. Point the new Link ID to go to the newly created view of your new document type.

c. Customize the JSP of the new Alert detail view to call the `getDetailHrefOptions()` JSP utility function with a Link ID parameter of the new document type.

d. Change the sequence of the new Alert detail view so that the new view becomes the default Alert detail view. For details, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

e. In order to make all links point appropriately to your new view, follow the steps in "Incorporating Your View across the Application" on page 66.

f. In order to retrieve the Document Type of the specific order number, configure the Alert detail view to call another API (after the exception detail API) to retrieve the order details. Configure the API to ignore exception. The API is called for all exceptions with the order number as a parameter, but the API throws an error if the order number is void. Since the API is configured to ignore the exception, it is not reported to the user.

3.22 Working with Document Types and Demand Records

Sterling Supply Chain Applications provide consoles for viewing demands stemming from Sales Orders (document type 0001), Returns (document type 0003) and Purchase Order (document type 0005) detail view document types from the Inventory Console. The Demand detail screen contains hyperlinks to the documents that create the specified demand. When creating a new document type (and appropriate document type UI) that can create demand, you must extend the Demand list screen to add a link resource that enables the user to navigate to the detail screen for your new document type.

In order to view any custom document types that may cause demand records, the Demand detail view must be extended to correctly display the details of the demand. The Demand detail screen requires an additional link resource for the new document type. After creating a new UI for your document type, extend the view as described below.

To extended demand list view for document types:

1. Extend the detail view (YIMD215) by following the directions in "Customizing a Detail View" on page 52.

2. Add a link resource under the custom inner panel that was copied from the YIMD215I01 inner panel. The link resource should have a postfix value containing the document type code of your new

document type and it should point to the view ID of the detail view for your new document type.

3. Set the Resource ID sequence so that the customized view is before that of the view defined by Sterling Supply Chain Applications.

3.23 Configuring Actions and Enabling Custom Transactions

Depending on your business processes, your pipeline may require an additional step that results in a status change of an entity. For example, you may want to add a security check to your order's lifecycle so that a customer service representative can manually authorize an order before shipping it.

Any custom status change transaction configured within the Scenario Modeling Configurator can be configured as an Action in the detail view of an entity. You can configure the Action to invoke the custom transaction directly, or you can invoke the custom transaction through a new custom view that permits the user to manually confirm individual transactions, as described below.

To create an action from an inner panel:

1. From the Scenario Modeling Configurator, configure a new transaction in the pipeline, along with the pick up and drop statuses that meet your requirements.

2. Under the inner panel of the custom detail view, create an Action resource that calls the `yfcShowDetailPopupWithParams()` JavaScript function. For more information on passing parameters to detail views, see "yfcShowDetailPopupWithParams" on page 689.

3. Make sure that the new transaction ID (created in Step 1) and the view resource ID specified in Table 3–5 are passes as input to the JavaScript function.

Table 3–5   Status Change Resource IDs

For This Entity...	Use This Resource ID...
Shipment	YOMD770
Load	YDMD280

Table 3–5   Status Change Resource IDs

For This Entity...	Use This Resource ID...
Order	YOMD390
Purchase Order	YOMD3390
Returns	YOMD512

Example 3–7   Status Change Action for Authorizing Returns

The following example shows the status change action for authorizing returns.

3.24 XML Binding

The input and output of APIs is in the form of an XML document. Using XML documents enables the Sterling Presentation Framework to provide an XML binding mechanism by which a developer can form the input necessary to pass to an API and populate a screen with its output.

The binding logic is based on using the name attribute of input fields to map on to an XML attribute. The actual HTML string that needs to be formed is returned by various HTML tag-specific JSP functions and JSP

tags that have been provided for that purpose. The HTML that is rendered contains the name attribute set to the appropriate XML path. When data is posted to the server, the servlet provided by the Sterling Service Definition Framework captures the request and forms an XML document out of the data in the input and XML path contained in the name attribute. The XML document is then passed as input to the API that has been configured for the action being performed.

For example, you may bind an input box to the ShortDescription attribute of the getItemList() API in an Item search view.

```
<tr>
    <td class="searchlabel" ><yfc:i18n>Short_Description</yfc:i18n></td>
</tr>
<tr>
    <td nowrap="true" class="searchcriteriacell" >
        <input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/Item/PrimaryInformation/@ShortDescription") %> />
    </td>
</tr>
```

After the JSP functions and JSP tags are resolved, the HTML is formed as follows:

```
<tr>
    <td class="searchlabel" >Short Description</td>
</tr>
<tr>
    <td nowrap="true" class="searchcriteriacell" >
        <input type="text" class="unprotectedinput"
name="xml:/Item/PrimaryInformation/@ShortDescription" value=""/>
    </td>
</tr>
```

Note that this example does not show all of the custom attributes returned by the JSP functions. It only shows the ones that are relevant to this topic.

In another example, the user enters Telephone in the input box. When the data is posted to the server, the Sterling Presentation Framework forms the following XML document based on the name and the value of the input box.

```
<Item>
<PrimaryInformation ShortDescription="Telephone"/>
</Item>
```

Since the Sterling Presentation Framework parses the binding string to form the XML, the binding string must follow the syntax below.

3.24.1 XML Data Binding Syntax

APIs are called with the input XML that is bound in the screen, and that XML binding should match the output of the API.

XML Binding Syntax

xml:NameSpace:/Root/Child@Attribute

The following examples show correctly structured syntax:

```
xml:/Order/PersonInfoShipTo/@Name
xml:Order:/Order/PersonInfoShipTo/@Name
```

The following examples show incorrectly structured syntax:

```
xml:/@Name
xml:/Order
xml:Order:/Order
```

XML Binding Parameters

Where the parameters are used as follows:

xml: - used literally

NameSpace - namespace containing the XML to which this binding applies. If not specified, it is take to be the root node of the path that follows. Namespace should only be included when binding to common codes.

:/Root/Child@Attribute - XML path of the attribute. If the attribute is the root node itself, specify the syntax as /Root@Attribute. *Root* represents the root node name in the XML to which the binding applies. *Child* represents the child element node name. This rule applies to any level of depth.

This function parses the binding string, searches for the at ("@") character, and returns the string following the at ("@") character.

3.24.2 Special XML Binding Considerations

The XML binding for an input field on the screen enables you to uniquely bind a field to a single attribute in the XML document. The XML binding is used as the `name` of the HTML input object. A binding consists of the complete XML path and attribute name of the target attribute. Given this fact, it is not immediately obvious how to bind input boxes to XML attributes that exist in an XML list. For example, given the following XML:

```
<Order>
<OrderLines>
<OrderLine OrderLineKey="1000001" ShipNode="ShipNode1"/>
<OrderLine OrderLineKey="1000002" ShipNode="ShipNode2"/>
</OrderLines>
</Order>
```

3.24.3 XML Binding for Multiple Element Names

The general rule for XML binding consists of using the full path and attribute name. However, this may result in multiple input objects in the JSP with the same name. Input objects with the same name are posted as an array of objects and not be posted to the API.

To uniquely identify each input as being part of a specific XML element, you can add a postfix that contains an underscore ("_") plus a counter after the repeating element name.

For example, the binding of each ship node field on the list of order lines should be `xml:/Order/OrderLines/OrderLine/@ShipNode`. If you require a screen that contains has a list of order lines with ship node editable on each line, you can use a special XML binding convention to handle this scenario.

The repeating element is OrderLine. For each ship node input object, the special postfix is added for each line. The result is two unique XML bindings: `xml:/Order/OrderLines/OrderLine_1/@ShipNode` and `xml:/Order/OrderLines/OrderLine_2/@ShipNode`. When this data is posted, all XML bindings containing the same special postfix are combined into the same XML element in the API input.

To make using this special postfix XML binding easier, the loopXML JSP tag provides a JSP variable that contains a unique counter for each individual loop. See "loopXML" on page 643. This JSP variable that is available inside the loopXML JSP tag is the ID attribute specified in the

loopXML tag plus the literal Counter. For example, use the following loopXML in your JSP:

```
<yfc:loopXML name="Order" binding="xml:/Order/OrderLines/@OrderLine"
id="OrderLine">
```

This makes the OrderLineCounter JSP variable available for use inside of the input XML bindings. For example:

```
<input type="text" <%=yfsGetTextOptions("xml:/Order/OrderLines/OrderLine_" +
OrderLineCounter + "/@ShipNode", "xml:/OrderLine/@ShipNode",
"xml:/OrderLine/AllowedModifications")%>/>
```

3.25 API Input

When customizing the user interface, you must ensure that the correct input is passed to the APIs you use. The primary way to retrieve data or perform updates in the user interface is through APIs. Making sure the right input is passed to these APIs is an important task for user interface customizations.

Sterling Supply Chain Applications provide various mechanisms for passing input to APIs through the user interface. Which mechanisms you should use, and in what combination, depends on the type of screen and type of API being called.

This section describes the features and advantages of the following mechanisms for passing data to APIs:

- Input namespace
- Entity key
- Dynamic attributes

Input Namespace

In many cases, data from fields on the UI must be passed directly to the API. A simple example of this is any detail screen that has editable input fields. These fields must be passed to a save API in order to update the entered data in the application's database. This save API takes a specific XML structure as input. The fields on the detail screen should have XML binding that matches the input to the API.

In the Resource Configurator, the Input Namespace field in the action resource should be set appropriately to ensure that the correct data from

the console is passed to the save API. See the *Sterling Supply Chain Applications Platform Configuration Guide.*

Since all of the input fields on the screen have XML bindings for the input to the same API, they all have the same XML root element name in the binding. This root element name of the XML bindings is known as the *namespace* of that input field. Therefore, when that save action is executed on the user interface, an XML is formed from all of the input fields containing the namespace specified for that action in the Resource Configurator. This XML is then passed to the API configured under that save action. For more information on XML binding, see "XML Binding" on page 92.

Another place where input namespace is frequently used is for user interface screens that have search criteria that must be passed to list APIs that fetch data based on the search criteria entered. All the search criteria fields should have XML binding matching the input to the list API. The root element name of this XML is the namespace that should be specified for the list view that is shown when the user executes the search.

Entity Key
Frequently detail screens have API calling actions that only require the primary key of the current entity to be passed as the input. When a detail view is brought up in the user interface, there is always at least one entity key passed to the detail view. This entity key consists of an XML structure containing the attributes that uniquely identify that entity. For example, the entity key of the order entity always contains an attribute for order header key. This entity key is automatically passed to the detail API of that entity. When this type of action is configured on a detail view, there aren't any other steps required to ensure the right input is passed to the API.

Dynamic Attributes
Sometimes the input expected to be passed to an API is not available through an input namespace or entity key. In these cases, using dynamic attributes may be applicable. All API resources configured in the Resource Configurator have an input field whose purpose is to provide the ability to specify dynamic attributes. The value for this field should be a valid XML structure using elements and attributes. The XML structure

specified here must match the exact input XML structure accepted by the called API.

One of the most common examples of using dynamic attributes is when a specific inner panel that must call multiple APIs in order to retrieve all the data it needs to display. For example, an inner panel of a detail view of the order entity might require calling the `getOrderDetails()` (the standard detail API), and the `getCommonCodeList()` API to retrieve some data for some combo box on the screen. Since common codes are stored at the rule-set level, it is mandatory to make sure the correct rule set for that order is passed as input to the `getCommonCodeList()` API. The `getOrderDetails()` API, given the correct output template, returns the rule set key for the order. This rule set key can be passed to the `getCommonCodeList()` API by referring to the output of the `getOrderDetails()` API as a dynamic attribute in the input XML of the `getCommonCodeList()`API.

For example, the `getOrderDetails()` API returns:

```
<Order OrderHeaderKey="…" OrderNo="…" RulesetKey="…"/>
```

The Input field of the API resource definition for `getCommonCodeList()`, should be:

```
<CommonCode CodeType="ORDER_TYPE" RulesetKey="xml:/Order/@RulesetKey"/>
```

Notice that the value of the RulesetKey attribute is set to an XML binding that refers to the output of the `getOrderDetails()` function. The exact same rules of XML binding that apply when binding inputs inside of a JSP also apply when using dynamic attributes. This example also shows another way dynamic attributes can be used. The CodeType attribute is also specified in the input field in the API resource definition. Here, the value of the attribute is simply set to the static value "ORDER_TYPE". This attribute and value are *always* passed to this API when it is called. Non-changing input values can be specified in this way.

Another possible value that you can use for a dynamic attribute for and API defined under a detail view is any attribute of the current entity key XML. This is useful when the input to pass to an API does not have the same XML structure of the entity key XML that has been formed in the detail screen. The XML for the current entity is available in a special namespace called SelectionKeyName.

The different mechanisms for specifying API input can be combined. For example, it may be necessary to pass the entity key AND some dynamic attribute to an API configured under a detail view action. Since passing the entity key happens automatically, you can still specify an Input under that API with the correct XML structure. Note that the XML structure of the key should match the XML structure of the input field. There are other special namespaces available for use in dynamic attributes. For more information, see "Available Dynamic Attribute Namespaces" on page 99.

3.26 Available Dynamic Attribute Namespaces

Sterling Supply Chain Applications have the following special namespaces available for use in dynamic attributes:

CommonFields - This namespace is only available when using the common_fields JSP. The attributes that are available depend on how the JSP is used.

CurrentUser - This namespace contains the details about the current logged in user using the getUserDetails() API. The exact XML available is:

```
<User Activateflag="" BillingaddressKey="" BusinessKey="" ContactaddressKey=""
Createprogid="" Createts="" Createuserid="" CreatorOrganizationKey=""
Imagefile="" Localecode="" Loginid="" Longdesc="" MenuId="" Modifyprogid=""
Modifyts="" Modifyuserid="" NoteKey="" OrganizationKey="" ParentUserKey=""
Password="" PreferenceKey="" Pwdlastchangedon="" Theme="" UserKey=""
UsergroupKey="" Username="" Usertype=""/>
```

CurrentEnterprise - If the current user belongs to an organization that is an enterprise, this namespace contains the details about that enterprise. If the current user belongs to an organization that is not an enterprise but participates in an enterprise, this namespace contains the details about the primary enterprise of the current organization. The details for the organization are retrieved from the getOrganizationHierarchy() API. The exact XML available is:

```
<Organization AccountWithHub="" AuthorityType="" BillingAddressKey=""
CatalogOrganizationCode="" CollectExternalThroughAr="" ContactAddressKey=""
CorporateAddressKey="" Createprogid="" Createts="" Createuserid=""
CreatorOrganizationKey="" DefaultDistributionRuleId="" DefaultPaymentRuleId=""
DunsNumber="" InterfaceTime="" InventoryKeptExternally=""
InventoryOrganizationCode="" InventoryPublished="" IsHubOrganization=""
```

```
IsSourcingKept="" IssuingAuthority="" ItemXrefRule="" LocaleCode=""
MerchantId="" Modifyprogid="" Modifyts="" Modifyuserid="" OrganizationCode=""
OrganizationKey="" OrganizationName="" ParentOrganizationCode=""
PaymentProcessingReqd="" PrimaryEnterpriseKey="" PrimarySicCode="" PrimaryUrl=""
RequiresChainedOrder="" RequiresChangeRequest="" RulesetKey="" TaxExemptFlag=""
TaxExemptionCertificate="" TaxJurisdiction="" TaxpayerId="" XrefAliasType=""
XrefOrganizationCode="">
```

CurrentOrganization - This namespace contains the details of the organization of the current logged in user using the `getOrganizationHierarchy()` API. The exact XML available is the same that is available under the CurrentEnterprise namespace.

SelectionKeyName - This namespace contains the XML that is bound to the currently active key of the current entity. Typically a list screen forms a XML key and associates that key to the checkbox (or hyperlink) which is used to navigate to the detail screen. This key is known as the current selected entity key. A detail view uses this key to call the detail API for that entity. For more details on this namespace, see

3.27 Posting Data to an API

By default, the data found in the editable components of a screen is not sent automatically to a save API, even if the input fields are bound to some XML. The data is only posted if the user has changed it on the screen.

However, sometimes it is necessary to pass some or all of the data in editable components, even if it did not change. For example, screens in which you are defaulting the input boxes to some default value in the JSP require all data to be posted.

If the user does not change the data in the input box, you still want the value to be passed to the API. Therefore, there is a mechanism that identifies that all data in the editable components is passed to the API for the entire JSP regardless of what was actually changed in the user interface. The following code example illustrates how to accomplish this.

```
<script language="Javascript">
IgnoreChangeNames();
</script>
```

Note that this code typically is located at the beginning of a JSP immediately following the JSP include statements.

3.27.1 Data Types

Input boxes on an HTML page must conform to specific constraints regarding field size and data type. For example, the size of an input box should correspond to the type of data being gathered. Likewise, each input box requires validations that correspond to the type of data the field is designed to gather (such as numeric, date, string, and so forth). For example, string fields must prevent the user from entering data longer than a specific length. The attribute used for binding a text box to an API output also resolves the data type and related properties such as size, decimal digits, and so forth.

The Sterling Presentation Framework has two APIs, `getTextOptions()` and `yfsGetTextOptions()`, that permit you to not have to explicitly set these attributes for each field. When you use these APIs for input boxes, they automatically take care of the data type-related attributes and validations.

These attribute definition and mappings are contained in two files:

- yfsdatatypemap.xml - maps abstract data types to XML attribute names
- YFSDataTypes.xml - defines data types

3.27.1.1 Abstract Data Type Mappings

XML attributes are mapped to abstract data types. The mappings are contained in the `<YANTRA_HOME>/Applications/Foundation/template/api/yfsdatatypemap.xml` file.

For example, if a `Name` attribute contains the XML attribute `TaxBreakupKey`, and the `DataType` attribute contains the abstract data type `key`, this is defined in the `yfsdatatypemap.xml` file as follows:

```
<Attribute Name="TaxBreakupKey" DataType="Key" />
```

3.27.1.2 Abstract Data Type Definitions

The abstract data types define the database data type (`DATE`, `VARCHAR2`, and so forth), the database size, and so forth. The abstract data types

are defined in the `<YANTRA_ HOME>/Applications/Foundation/template/api/YFSDataTypes.xml` file.

Below are a few sample entries from the file:

```
<DataType Name='Address' Type='VARCHAR2' Size='70'  >
<UIType Size="30" UITableSize="30"/>
</DataType>
<DataType Name='Count' Type='NUMBER' Size='5'  NegativeAllowed="false"
ZeroAllowed="true">
<UIType Size="5" />
</DataType>
<DataType Name='TimeStamp' Type='DATETIME' Size='17'  />
<DataType Name='Date' Type='DATE' Size='7'>
<UIType Size="12" UITableSize="15"/>
</DataType>
<DataType Name='Quantity' Type='NUMBER' Size='10' NegativeAllowed="false"
ZeroAllowed="true">
<XMLTypeType="QUANTITY"/>
</DataType>
```

For a definition of the standard Sterling Supply Chain Applications abstract data types, see the `<YANTRA_ HOME>/Applications/Foundation/template/api/YFSDataTypes.xml` file.

For a list of attributes supported `YFSDataTypes.xml`, see "Data Type Reference" on page 702.

3.27.1.3 Data Type Determination

Sterling Supply Chain Applications use the `yfsdatatypemap.xml` file for data type determination. First Sterling Supply Chain Applications search the file to find the string specified for binding (including the `xml:` prefix). If the entire binding string is not found, Sterling Supply Chain Applications search the file for only the attribute part of the binding string.

The attribute must be able to be found in one of these two formats.

Then, using the definition in the `YFSDataTypes.xml` file, the Sterling Presentation Framework API forms an appropriate HTML string with custom attributes that are then used on the client side.

3.27.1.4 Data Type Validation

A JavaScript function on the client side executes when the page loads, then reads the custom attributes and sizes the input boxes appropriately. A validation function is attached to these input boxes through the `window.attachEvent()` function. The validation function also uses the custom attributes to perform data type validations.

3.27.1.5 Customizing the Data Types Files

You can extend the attributes available to you by adding your own XML attributes and abstract data types to the `YFSDataTypes.xml` file.

To extend the data type map XML file:

1. Create a new `<YANTRA_ HOME>/Applications/Foundation/template/api/extn/yfsdatatype map.xml` file.

2. Add an XML root node in the same way it appears in the `<YANTRA_ HOME>/Applications/Foundation/template/api/yfsdatatypemap.x ml` file.

3. Add any attributes that need to be mapped in the `datatypemap.xml` file.

To extend the data type XML file:

1. Create a new `<YANTRA_ HOME>/Applications/Foundation/template/api/extn/YFSDataType s.xml` file.

2. Add an XML root node in the same way it appears in the `<YANTRA_ HOME>/Applications/Foundation/template/api/YFSDataTypes.xml` file.

3. Add any differential values for the datatypes, including the following:

   • Add and define parameters for new datatypes

   • Modify parameters of existing datatypes

> **Note:** For existing datatypes, you can modify only the UI related attributes in the YFSDataTypes.xml file such as UI Size, and UITableSize.

> **Note:** You cannot resize the Date input fields within the Sterling Supply Chain Application Consoles across the board even if your date format is larger than the default date format used by Sterling Supply Chain Applications.

> **Note:** Sterling Supply Chain Applications reserve the `Type` attribute for internal use, and so you cannot override it. All other attributes can be overridden.

4. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide.*

3.28 Displaying Credit Card Number in a New Screen

Credit card number should be displayed only to users who have permissions to see them. Therefore, when you build a custom screen to display credit card number, use the following rules to ensure that this security is maintained:

- CurrentUser namespace contains the attribute ShowCreditCardInfo under the User node. This attribute is `true` if the current login user does have permission to see the credit card number and `false` if the current login user does not have the necessary permissions

- APIs that return credit card number normally return the encrypted credit card number. These APIs also return a DisplayCreditCardNo attribute that contains the last four digits of the credit card.

- Use the DisplayCreditCardNo attribute in conjunction with the showEncryptedCreditCardNo() JSP function to initially show the credit card number as asterisks (*) followed by the last four digits. See "showEncryptedCreditCardNo" on page 628.

- Form a hyperlink on the credit card number that displays only if the logged in user has permission to see decrypted credit card numbers. For example,

```
<% if (userHasDecryptedCreditCardPermissions()) {%>
    <yfc:makeXMLInput name="encryptedCCNoKey">
        <yfc:makeXMLKey
binding="xml:/GetDecryptedCreditCardNumber/@EncryptedCCNo"
value="xml:/PaymentMethod/@CreditCardNo"/>
    </yfc:makeXMLInput>
    <td class="protectedtext">
        <a
<%=getDetailHrefOptions(decryptedCreditCardLink,
getParameter("encryptedCCNoKey"),"")%>>

<%=showEncryptedCreditCardNo(resolveValue("xml:/PaymentMeth
od/@DisplayCreditCardNo"))%>
        </a>
    </td>
<% } else { %>
    <td class="protectedtext">

<%=showEncryptedCreditCardNo(resolveValue("xml:/PaymentMeth
od/@DisplayCreditCardNo"))%> 
        <yfc:getXMLValue
binding="xml:/PaymentMethod/@DisplayCreditCardNo"/> 
    </td>
<% } %>
```

- Then create a popup window that opens when the hyperlink is clicked.

- Call getDecryptedCreditCardNumber() in the popup window to decrypt the credit card, passing the DisplayFlag attribute as true if the current login user has permissions and false if the current login user does not have permissions.

- Use the output of getDecryptedCreditCardNumber() to display the decrypted credit card number on the screen.

When you configure the `getDecryptedCreditCardNumber()` API for your screen through the Sterling Supply Chain Applications Configurator, you must specify a dynamic input so that the DisplayFlag attribute is passed to the API, based on current user's permissions. Here is an example of how you could specify the Input field in the Sterling Supply Chain Applications Configurator:

```
<GetDecryptedCreditCardNumber
DisplayFlag="xml:CurrentUser:/User/@ShowCreditCardInfo"
EncryptedCCNo="xml:/Order/PaymentMethods/PaymentMethod/@CreditCardNo"/>
```

And specify the Template field according to the following example:

```
<GetDecryptedCreditCardNumber DecryptedCCNo=""/>
```

3.28.1 Displaying Multiple Credit Card Numbers

When displaying credit card numbers in a list, you might choose to display the DisplayCreditCardNo attribute, which is returned by the APIs that output CreditCardNo.

To append asterisks to the credit card number returned by the API, use the DisplayCreditCardNo attribute and the `showEncryptedCreditCardNo()` method.

Displaying a list of decrypted credit card numbers in a list involves calling `getDecryptedCreditCardNumber()` in a loop for each row. This can be an expensive operation, so you may want to display a list of encrypted credit card numbers (shown as `**********1234`) by using the DisplayCreditCardNo attribute. All APIs that output CreditCardNo return this attribute. Then link the encrypted credit card numbers to a popup window that displays a specified credit card number in a decrypted format.

3.29 Incorporating JSP Modifications in Sterling Supply Chain Applications EAR

After you customize the console user interface, you need to include these changes into the EAR file to re-deploy Sterling Supply Chain Applications to use your modifications.

To incorporate your JSP file modifications after customizing the interface as specified in this chapter, run the following script from the `<YFS_HOME>` directory, which creates an UI extension jar:

`ant -buildfile bin/buildXX.xml ui-extn`, where `XX` represents either WLS or WS.

where <YFS_HOME> refers to the `<YANTRA_HOME>/Runtime` directory.

You can either create a new EAR or update the existing EAR after extending the UI. For more information on the various targets for building the EAR see the *Sterling Supply Chain Applications Installation Guide*.

4

Customizing the Configurator Swing Interface

The Sterling Presentation Framework extensibility of Sterling Supply Chain Applications enables you to change the way information is rendered, or displayed, in the Sterling Supply Chain Applications Configurator, without changing the way it functions.

This chapter describes the Sterling Supply Chain Applications Configurator user interface and walks you through the tasks necessary for customizing it to suit your business needs. Each task is accomplished through a combination of configuration changes through the Sterling Supply Chain Applications Configurator and Java swing code changes.

4.1 Swing User Interface Extensibility

The main purpose of user interface extensibility is to enable any database extensions to be integrated into the graphical user interface.

Extensibility includes the following modifications:

- Adding any icons (or buttons) and labels
- Adding any text fields and checkboxes
- Hiding any non-mandatory components
- Reorganizing the components that are displayed on-screen

> **Note:** If you extend the Swing user interface, when you install upgrades and services packs you need to read the upgrade documentation and carefully reconcile changes that have taken place in the default screens of Sterling Supply Chain Applications. Some changes may be mandatory while some may not be.

You may modify the following types of screens:

- Search screens
- Detail screens
- List screens

The explorer screens (screens that contain only a tree) are not extensible.

4.1.1 Extending Organization and Item Detail Screens

When you extend a table used by the Item Details screen or the Organization Details screen, this adds a popup action icon to the main screen. This icon enables access to a popup screen that contains all of the relevant extended fields.

For example, when you extend the organization table, a popup access icon is added to the default organization screen. When the user clicks this icon, a popup screen displays all of the extended attributes that are relevant to this table.

Typically, extended attributes on the popup screen appear as text input boxes. In the case of the Item Detail screen, any item attribute that is specified as a classification displays as a classification lookup. This lookup enables the user to select any classification value that has been configured as described in the *Sterling Product Management Configuration Guide*.

Fields on the popup screen can be grouped together. These groups are displayed in alphabetical order on the screen. Likewise, the fields within each group are displayed in alphabetical order.

To add extended attributes to a popup:

1. Edit the `Extensions.xml` file in `<YANTRA_ HOME>/Applications/Foundation/database/entities/extensions` directory and specify a DataType attribute for your new column. The new attribute should be a new data type that has not been defined previously in the YFSDataTypes.xml file.

2. Edit your `YFSDataTypes.xml` file in `<YANTRA_ HOME>/Applications/Foundation/template/api/extn` directory and add an entry for your new data type using the following attributes:

   - **DisplayInUI** - Required. Specifies whether the field should automatically display in the new popup screen.

   - **DisplayGroup** - Optional. Groups the display of extended fields on the new popup screen. When this field is specified, all extended fields with the same DisplayGroup attribute are grouped together and displayed in alphabetical order beneath a title that is set to the value of the DisplayGroup attribute.

   For example, attributes for the newly added ExtnMyField data type are specified as follows:

   ```
   <DataType Name='ExtnMyField' Type='CHAR' Size='35'>
   <UIType Size="30" UITableSize="30" DisplayInUI="true" DisplayGroup="My_
   Group"/>
   </DataType>
   ```

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4.1.2 Extending Search and Detail Screens

The Sterling Supply Chain Applications Configurator menu structure contains a hierarchical collection of the following types of items:

- Menu – Contains child menu items

- Resource – Contains no child menu items, and instead, points to a resource

The Sterling Supply Chain Applications Configurator menu structure cannot be modified.

> **Note:** You need Net Beans 3.2 IDE to extend search and detail screens in the Configurator.

Each screen within the Sterling Supply Chain Applications Configurator is associated with a Form Class and a Java Behavior Class. The Form Class is responsible for painting the controls on the screen and the Behavior Class is responsible for populating data in the screen and responding to events that occur in the screen, such as choosing Save.

When navigation to a screen within the Sterling Supply Chain Applications Configurator, the Form Class of the corresponding screen is loaded and the Behavior Class populates the data in the screen.

The Sterling Supply Chain Applications Configurator screens are defined in an XML file. This file contains the unique screen ID, Form Class, and Behavior Class for each screen. This file must be extended in order to extract the Sterling Supply Chain Applications Configurator screens as described in the following steps:

To extend a search or detail screen:

1. From the Sterling Supply Chain Applications Configurator, navigate to the screen that you want to extend.

2. After the screen loads, press CTRL-M, which displays the window with the Form Name (which is the resource ID for the screen), the Form Class name, and XML data information.

3. Note the Form Name (resource ID) and Form Class Name.

4. The `<YANTRA_ HOME>/Applications/Foundation/documentation/code_ examples/configuisrc/yantrauisrc.jar` file contains the source code corresponding to all `Form` classes. There is a corresponding `.form` file and a `.java` file. The `.form` file is used by NetBeans and is required only if you use the NetBeans 3.2 IDE.

5. Copy the `.java` and `.form` files corresponding to the Form Class Name that you had noted into your own directory structure. The copy you make should have a different class name. Make sure you do not copy it anywhere under com.yantra because that is reserved strictly for products of Sterling Supply Chain Applications.

6. Add the following JAR files to the CLASSPATH. This can be done in NetBeans 3.2 by mounting the JAR file.

- jgo.jar
- ycpui.jar
- ycmui.jar
- yifui.jar
- yfcui.jar
- xercesImpl.jar

You need to do this in order to compile the Java file.

7. Add the package name to the top of the form. Put the appropriate class name in the code (should be the same Java class that you originally copied as the file name created in step 5).

The copied Java class must extend the original.

8. Set the Variables Modifier option in Net Beans to `public`. The default value is `private`. This option can usually be found in Tools > Options > Form Objects > Expert Tab.

9. Remove `super.init()` from the `init()` function.

10. At the end of the `init()` function, add the following line:

```
checkVars();
```

11. Make the necessary changes to the new form. To set the properties of the new controls, see Table 4–1, "XML Binding". Only the following changes are permitted:

- Rearranging any components on the user interface
- Hiding any non-mandatory components
- Adding any buttons and labels
- Adding any text fields and checkboxes

12. Compile the `.java` file, create a JAR file that contains only the `.class` file, and put it in the `<YANTRA_HOME>/Applications/Foundation/extn/ui/` directory and the `<YANTRA_HOME>/Applications/Foundation/webpages/yfscommon/` directory. The JAR file name should be `yfsextn.jar`.

13. Rename the `<YANTRA_ HOME>/Applications/Foundation/template/configapi/extn_ application.xml.sample` file to "extn_application.xml".

14. Edit the `extn_application.xml` file to include the Form Name (the resource ID that you noted in step 3) and the Override Form Class Name (the complete path of the new class name that will override the existing Sterling Supply Chain Applications class).

15. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

16. From the `<YFS_HOME>` directory, run the following script, which creates an application EAR file:

    `ant –buildfile bin/buildXX.xml ui-extn create-ear`, where XX represents either WLS or WS.

    where `<YFS_HOME>` refers to the `<YANTRA_HOME>/Runtime` directory.

    You can either create a new EAR or update the existing EAR after extending the UI. For more information on the various targets for building the EAR see the *Sterling Supply Chain Applications Installation Guide*.

4.1.2.1 XML Binding

All the forms in the Swing user interface follow the Model-View-Controller (MVC) paradigm. The form itself acts as the View, so it only has presentation logic. All the business logic is in a separate Controller class. The model for every form is an XML-DOM Element.

To further simplify the presentation logic, the core classes in <yfs> support a form of XMLBinding of different types of controls to the DOM model. This enables the form designer to bind the different controls on a form to different parts of the DOM. At run-time the infrastructure keeps the DOM and the controls synchronized. When the DOM is changed the changes are reflected on the control, and the reverse is true as well.

The following controls can be bound:

• Javax.swing.JTextComponent (and any subclasses)

• Javax.swing.JTable

The binding semantics control are described in detail below.

4.1.2.2 Binding Common to All Controls

Given an input XML, parts of the XML can be bound to controls based on an XMLBindingString. Each string is evaluated in XSL syntax and the first match is used as the value of the binding.

Assume the bound XML is the following:

```
<Order OrderNo='23' OrderDate='20010101' >
<ShipToAddress City='Nashua' />
</Order>
```

Table 4–1 illustrates the XMLBinding.

Table 4–1   XML Binding

XML Binding String	Example Value
Xml:/Order/@OrderNo	23
Xml:/Order/@OrderDate	20010101
Xml:/Order/ShipToAddress/@City	Nashua
Xml:/Order/ShipToAddress	Evaluates to the ShipToAddress Node

4.1.2.3 Name Property

Every Swing control has a bean property name that can be set by using the setName(String) function. The value of this property should be an XMLBindingString that specifies the XMLData that is to be bound to that control.

4.1.2.4 Binding for JText Field

In addition to the Name property, the following TextField properties can be set for any text field.

Table 4–2   JTextField Properties

Property	Syntax
Data Type: Only Integer, Date and String are supported.	txtField1.putClientProperty("YFCXMLBinding.dataType", "Integer")
	txtField1.putClientProperty("YFCXMLBinding.dataType", "Date")
	txtField1.putClientProperty("YFCXMLBinding.dataType", "String")
Associated Label: This is the JLabel associated with the text field	txtField1.putClientProperty("YFCXMLBinding.assocaitedLabel", "lblField1")
Mandatory: If Mandatory property is set to "true" and the field is left blank, the label associated with the text field changes color. This is triggered on the lostfocus event of the text field.	txtField1.putClientProperty("YFCXMLBinding.isMandatory", "true")

4.2 Extending the List Screens

You can add, remove, and re-arrange columns in the list screen within the Configurator using the following steps:

To extend list screens:

1. Copy `<YANTRA_HOME>/Applications/Foundation/template/api/genericscreens_modifications.xml.sample` to `<YANTRA_HOME>/Applications/Foundation/template/api/extn/genericscreens_modifications.xml`.

   For an example of the format of this file, see Example 4–1.

Example 4–1   List Modification XML File Structure

```
<ScreenModifications>
   <resourceId of the list form you want to extend>
      <ListInfo>
         <Absolute>
            <List AttributeName="" ColumnTitle="" DataType=""/>
```

```
            <List AttributeName="" ColumnTitle="" DataType=""/>
        </Absolute>
        <Add>
            <List AttributeName="" ColumnTitle="" DataType=""/>
        </Add>
        <Remove>
            <List AttributeName="" ColumnTitle="" DataType=""/>
        </Remove>
    </ListInfo>
  </resourceId of the list form you want to extend>
</ScreenModifications>
```

2. Edit the `genericscreens_modifications.xml` file and insert entries for the resource IDs that correspond with the list screens that you want to extend.

> **Tip:** In order to determine the Resource ID for a specific list screen, use the following steps:
>
> **a.** From within the Configurator, choose the User Group screen.
>
> **b.** Choose the Permission tab to display a hierarchical view of all permissions.
>
> **c.** Search the permission tree to find the list screen that you want to extend.
>
> **d.** Hover your mouse over the node to display the Resource ID using the tool tip.

3. Enter the ListInfo element attributes, using the descriptions listed in Table 4–3, "Elements in the List Modification XML File".

Table 4–3   Elements in the List Modification XML File

ListInfo Element	Purpose
Absolute	Replaces the current columns of a list with the new columns you specify in the List elements. Overrides Add and Remove.
Add	Adds one or more columns to the list screen. Add and Remove are mutually exclusive with Absolute.
Remove	Removes one or more columns from the list screen. Add and Remove are mutually exclusive with Absolute.

Table 4–3   Elements in the List Modification XML File

ListInfo Element	Purpose
List	The List element is a sub-element of the Absolute, Add, or Remove element. Adds the following attributes (which must be specified): • AttributeName - The name to display in the list attribute. • ColumnTitle - The title to display in the list in the attribute. • DataType - The data type to display in the new column. Must be a valid DataType Name within the `YFSDataTypes.xml` file.

4. Open the `<YANTRA_ HOME>/Applications/Foundation/template/api/YFSDataTypes.xml` file to determine which data type you should specify in the DataType attribute.

5. If the data type you need is not in the file, create a new data type, filling in the element attributes using the descriptions listed in Table 4–4, "Elements in the YFSDataTypes.xml File".

Table 4–4   Elements in the YFSDataTypes.xml File
Table 1:

DataType Element	Purpose
Name	Identifier of the DataType. Must exactly match the List element DataType attribute used in the `genericscreens_modifications.xml` file.
Type	Adds the following attributes (of which, Type and Size must be specified): • CHAR - Attribute Size • DATE - Attribute Size • NUMBER - Attributes Size, DecimalDigits (optional), NegativeAllowed (optional), ZeroAllowed (optional) • VARCHAR2 - Attribute Size

6. Save your changes to the genericscreens_modifications.xml file and close it.

7. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

8. Select the refresh cache icon that fits your needs as follows:

   – If you want to update one entity and its child resources - Select the specific entity and choose the ◈ *Refresh Entity Cache* icon

   – If you want to update all resources - Choose the ⟳ *Refresh Cache* icon

9. Run Sterling Supply Chain Applications again to test your changes.

4.3 Creating and Modifying User Themes

User themes determine the set of colors used for graphical user interface elements such as screens, labels, and table headers. For information on modify themes, see Chapter 3, "Customizing the Console JSP Interface".

4.4 Creating and Modifying Custom Error Codes

In the Sterling Supply Chain Applications Configurator, you can create error codes to be thrown for any exception specified in your custom code. The associated cause, action and description defined while creating the error code should be available in your user interface. Custom error code values should not contain any Sterling Supply Chain Applications reserved keywords. For a list of reserved keywords see Appendix A, "Special Characters Reference".

Custom error codes are also used for failure reasons thrown by the password validation user exits. These user exit exceptions are thrown when a user attempts to save the password changes in the console. A sample implementation of these user exits using Java's built-in MD5 routines are provided in <YANTRA_ HOME>/Applications/Foundation/documentation/code_ examples/pwcrypt directory. For more information on these user exits refer to *Sterling Supply Chain Applications Javadocs*.

For more information on creating or modifying custom error codes refer to the presentation component in the *Sterling Supply Chain Applications Platform Configuration Guide*.

4.5 Customizing Node Type Symbols for the Fulfillment Network Model

In the Fulfillment Network Model in the Sterling Supply Chain Applications Configurator, node types are displayed as symbols of various shapes, colors and sizes. Using the provided `extn_ mapmanager.xml.sample` file you can modify the look and feel of these node types on the Fulfillment Network Model. For more information on the Fulfillment Network Model, refer to the *Sterling Supply Chain Applications Platform Configuration Guide*.

To customize node type symbols:

1. Rename the `<YANTRA_ HOME>/Applications/Foundation/template/configapi/extn_ mapmanager.xml.sample` file to "extn_mapmanager.xml".

2. Edit the `extn_mapmanager.xml` file to include the node type you are customizing and the include the applicable information.

   Valid values for `ShapeType` are:

   - `Ellipse`
   - `Rectangle`
   - `RoundRectangle`
   - `Diamond`
   - `TriangleUp`
   - `TriangleDown`
   - `TriangleLeft`
   - `TriangleRight`
   - `Marker`

   For `Color` and `Selected Color`, specify any hex code or standard color name.

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4. Either create a new EAR or update the existing EAR after extending the UI. For more information on the various targets for building the EAR see the *Sterling Supply Chain Applications Installation Guide*.

5

Creating Custom Mobile User Interfaces

Sterling Supply Chain Applications enables you to develop and display a custom user interface for the mobile devices used in warehouse operations. This chapter describes the concepts and procedures for developing mobile device user interface applications.

> **Important:** When customizing the interface, copy the standard resources of Sterling Supply Chain Applications and then modify your copy, or create a completely new view. **Do not modify the standard resources of Sterling Supply Chain Applications.**

5.1 Before You Begin

Before developing the Sterling Supply Chain Applications mobile user interface, familiarize yourself with guidelines listed in this section in order to help ensure a successful experience and enable you to work more quickly and with fewer errors.

Required Prerequisite Concepts

Before beginning, you need to understand how to develop HTML, JSP, and XML components, how to use APIs, and how to use the Sterling Supply Chain Applications Console and Configurator user interfaces.

Note that the mobile device user interface differs from the Console user interface in the following ways:

- Mobile device screens use separate architecture for search and list views. If you need search view and list views functionality, model them as detail views.

- Mobile device screens can have only one detail view. Each detail view can contain only one inner panel.

- A mobile device inner panel cannot have any actions or icons.

The APIs return the data that needs to be displayed. For information about using APIs, see Chapter 8, "Programming Transactions". For information on functions specific to mobile devices that are used within the JSP files, see Appendix D, "Mobile User Interface Extensibility Reference".

Customizing the mobile device user interface is accomplished using the Sterling Supply Chain Applications Configurator user interface. For more information about this UI, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

Prepare for Smooth Upgrades and Easy Maintainability

- *Do not* change the resource definitions of any of the resources shipped as part of the standard default configuration. Either make a copy through the Sterling Supply Chain Applications Configurator and then change the copy or create your own new views.

- *Do not* change any of the JSP files, JavaScript files and icon JAR files that are supplied by Sterling Supply Chain Applications. If you do, your changes may be lost during upgrades.

- When creating new views, consider issues regarding ease of maintenance as well as ease of creation. When you create a new view, inner panel, and so forth, it is possible to link to the JSPs supplied by Sterling Supply Chain Applications. But in future releases, Sterling Supply Chain Applications may add more resources to these JSP, which means you must monitor software changes and update your configuration to account for these changes.

Build in Usability

Any new views you develop should look and behave like the product views, so before you begin developing, gain an understanding of how the default views behave. For more information on the basic product look and feel, see "Introducing the Screen Layout and Behavior" on page 35.

Prepare Your Development Environment

In order to start the customization process, you must perform prepare the development environment to accommodate for development and testing of the Sterling Supply Chain Applications mobile user interface changes. See "Understanding the Development Environment".

5.2 Planning Your Mobile Device Screens

Create a design document, including a prototype, as described in this section.

5.2.1 Design Guidelines

In order to optimize the display of data and execution of transactions, design simple screens and simple transactions, using the following rules:

- Avoid placing a lot of information into a small space. This ensures more rapid transaction time and enables the end user to parse data visually more quickly.

- Because of the reduced screen size, if you need to display a lot of data, the display of data may need to be altered to accommodate the amount of data. In this case, the data must be persisted from one screen to the next before finally being posted. For small screens, use the TEMPQ utilities to pass data between screens.

- The TempQ utilities enables you to persist and pass data from one screen to the next. For information on the TempQ utilities, please see Appendix D, "Mobile User Interface Extensibility Reference".

- Provide text on one line and the data field on another line to accommodate for internationalization requirements.

- Validate fields only when necessary in order to optimize transaction execution time.

- Choose fast or templatized APIs that return exactly the correct type of information needed in order to optimize transaction execution time. For information creating optimized API output templates see Section 8.2.9, "Best Practices for Creating Custom Output XML Templates" on page 471.

5.2.2 Mobile Device Screen Size Dimensions

Restrict the screen size to 8 lines X 24 (or 22) characters per line. This ensures that your custom screen will also display correctly on a VT220 terminal.

A mobile device screen can contain only table.

A mobile device screen can handle a combined total of the following hidden and displayed fields as described in Table 5–1, "Mobile Device Screen Size Specifications".

Table 5–1   Mobile Device Screen Size Specifications

Field	Maximum Size
text and hidden fields	15
labels and protected fields	15
command buttons	5

When the maximum allowed size for a given field is violated by a user, (for example, when a user enters 20 hidden fields), the following error message is displayed: "An error was encountered while running this program: Invalid procedure call or argument".

Draw a layout of each screen. For example, creating inventory screens requires an inventory inquiry screen and an inventory detail screen.

Figure 5–1   Inventory Inquiry Prototype

Screen 1:   **Screen 2:**
Inventory Inquiry Screen   **Inventory Detail Screen**

1	12345678901234567890
1	Inventory Inquiry
2	Enterprise [ ]
3	Item ID [ ]
4	Desc
5	Product Class [ ]
6	UOM [ ]
7	
8	
	[Back] [Next View] [Inquire]

2	12345678901234567890
1	Enterprise [ ]
2	Reference [ ]
3	Item ID [ ]
4	Desc
5	Product Class [ ]
6	UOM [ ]
7	Quantity 99999999999
8	
	[Back]

Note the use of fixed width font while drawing the screen layouts. The buttons are not counted while considering the 8-row limit.

Figure 5–2   Inventory Inquiry Screens

Screen 1:
Inventory Inquiry Screen
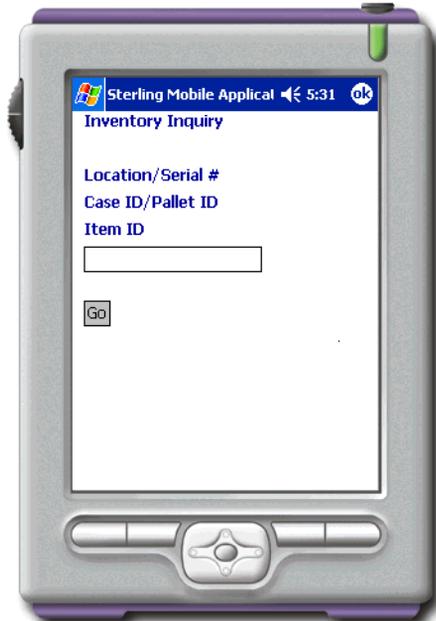
Screen 2:
Inventory Detail Screen
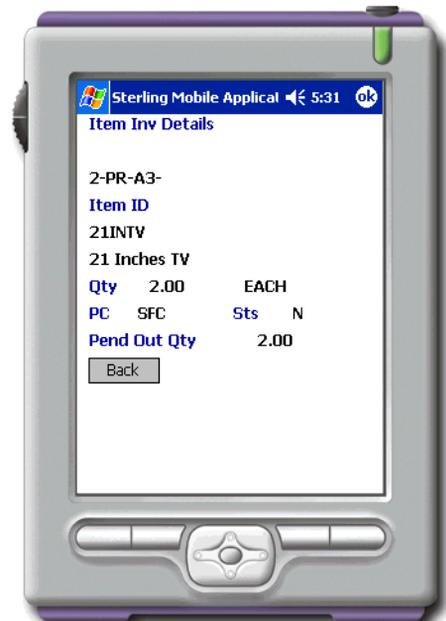
In this example, the getItemList() API fetches item details based on the information submitted on the Inventory Inquiry Search Screen, and getATP() API fetches inventory details for the item on the Inventory Inquiry Detail screen.

List the APIs that must be called when each screen is navigated to. This should include the entire API input that will be passed and the API template that will be used for filtering the API output, if applicable.

5.3 Creating Resources in the Configurator

Mobile device screens are composed of screen resources, such as an entity, a detail view, inner panels, and APIs. These resources define the screen look and feel, screen behavior, and screen flow.

To create custom mobile device resources:

Choose Applications > Platform > Presentation > Resources > Sterling Supply Chain Mobile Application > Entities as described in *Sterling Supply Chain Applications Platform Configuration Guide*.

Configure the resources described in Table 5–2, "Mobile Device Resources".

Table 5–2   Mobile Device Resources

Resource	Description
Screen Entity	Controls access to transactions. It also provides the starting point (JSP name).
Detail View	Mobile device screens are modeled as details views. A screen can have only one detail view. Each detail view can contain only one inner panel. A screen can contain only table.
Inner Panel	Mobile device inner panels cannot have any actions or icons.
APIs	Required APIs can be defined under the APIList for the innerpanel.

Example 5–1   Inventory Inquiry Configurator Resources

Creating Inventory Inquiry screens requires the following user interface resources:

- a screen entity called rfinventory

- detail view called rfinventoryD1

- inner panel called rfinventoryIP1

- `getItemDetails()` API called rfinventoryD1IP1API

These resources are detailed in the following tables.

Mobile Device Screen Entity Resources

Right click to create resources with the following values:

Figure 5–3   Mobile Device Screen Entity Resource
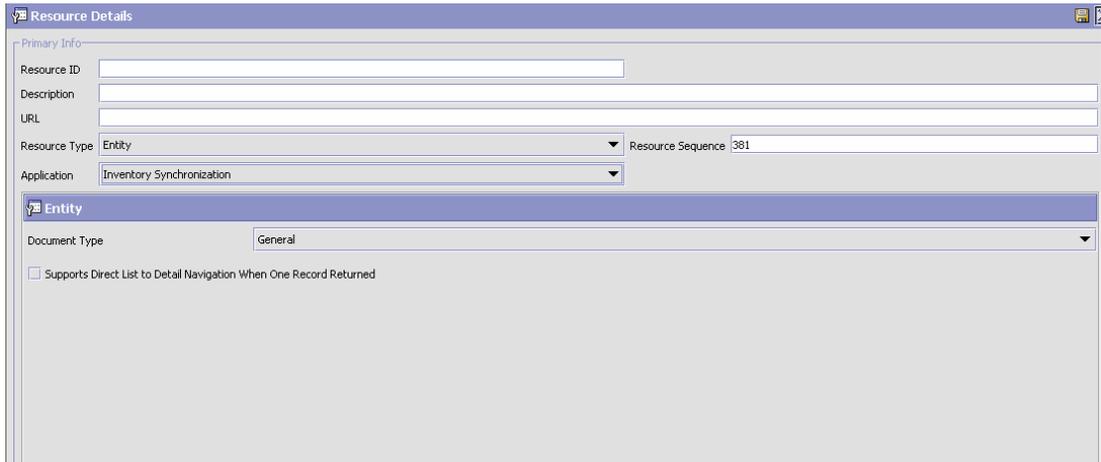
Table 5–3   Mobile Device Screen Entity Resource Values

Name	Value
Resource ID	rfinventory
Description	RF_Inventory
Resource Type	Entity
Resource Sequence	(Default Suggested Value)
Application	Warehouse Management
Document Type	General

Figure 5–4   Mobile Device Screen Detail View
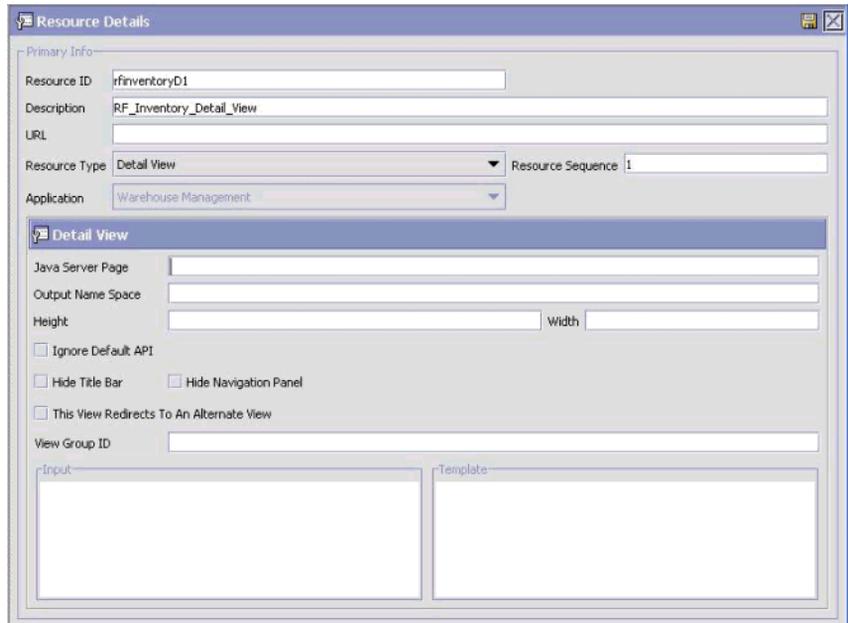
Table 5–4   Mobile Device Screen Detail View Resource Values

Name	Value
Resource ID	rfinventoryD1
Description	RF_Inventory_Detail_View
Resource Type	Detail View
Resource Sequence	(Default Suggested Value)
Application	Warehouse Management

Figure 5–5   Mobile Device Screen Inner Panel Resource
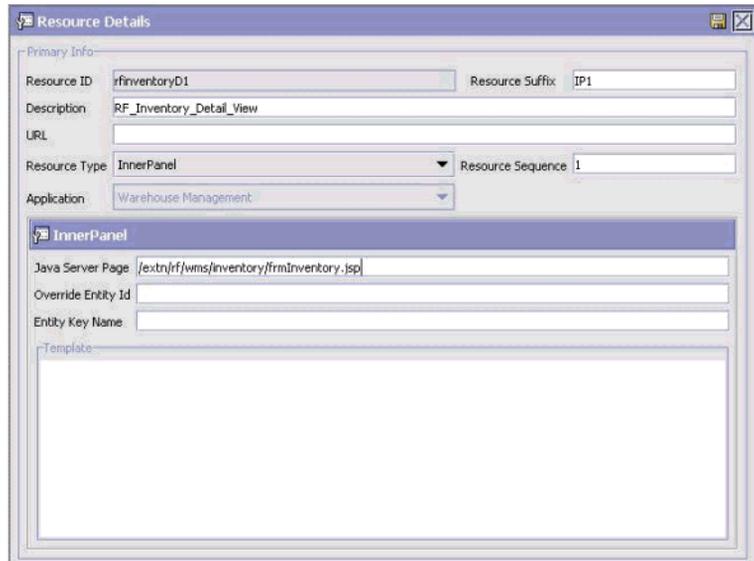
Table 5–5   Mobile Device Screen Inner Panel Resource Values

Name	Value
Resource ID	rfinventoryD1IP1
Description	RF_Inventory_Inner_Panel
Resource Type	Inner Panel
Resource Sequence	(Default Suggested Value)
Application	Warehouse Management
Java Server Page	/extn/rf/wms/inventory/frmInventory.jsp

Figure 5–6   Mobile Device Screen API Resource
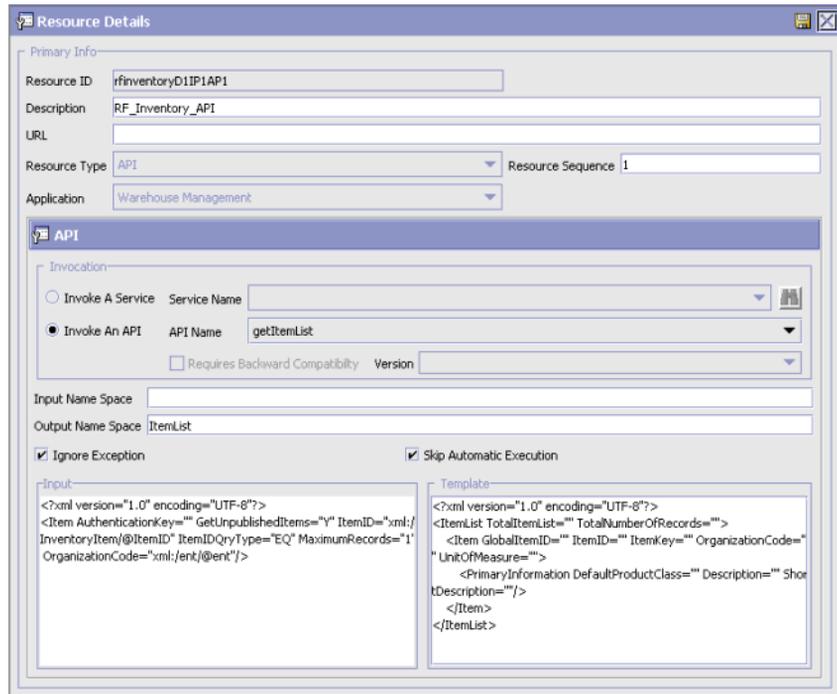
Table 5–6   Mobile Device Screen API Resource Values

Name	Value
Resource ID	rfinventoryD1IP1AP1
Description	RF_Inventory_API
Resource Type	API
Resource Sequence	(Default Suggested Value)
Application	Enter Warehouse Management.
Invoke an API	Select this radio button.
API Name	getItemList
Output Name Space	ItemList
Ignore Exception	Check this checkbox.

Table 5–6   Mobile Device Screen API Resource Values

Name	Value
Skip Automatic Execution	Always check this checkbox when defining a mobile device screen API resource.
Input	<?xml version="1.0" encoding="UTF-8"?><Item AuthenticationKey="" GetUnpublishedItems="Y" ItemID="xml:/InventoryItem/@ItemID" ItemIDQryType="EQ" MaximumRecords="1" OrganizationCode="xml:/ent/@ent"/>
Template	<?xml version="1.0" encoding="UTF-8"?><ItemList TotalItemList="" TotalNumberOfRecords=""> <Item GlobalItemID="" ItemID="" ItemKey="" OrganizationCode="" UnitOfMeasure=""> <PrimaryInformation DefaultProductClass="" Description="" ShortDescription=""/> </Item></ItemList>

5.4 Adding a Menu Entry

To create a mobile device menu entry

1. Create a menu entry by using the Menu Configurator. On saving the screen, a new Menu Group called "Custom Menu" is created.
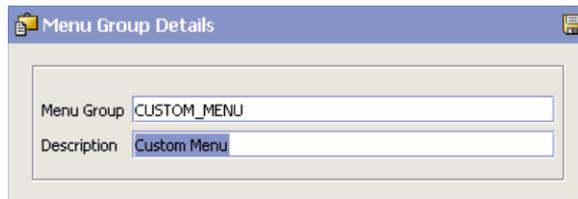
2. Right click on Custom Menu > Default_Yantra_Mobile_Menu > Create New Menu Item and choose Details.
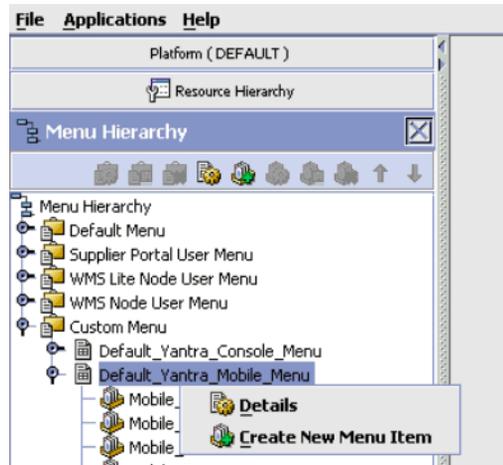
3. Add a new Menu Item with the description Mobile_Inventory_Inquiry the Resource ID `rfinventory` and the menu sequence as suggested.

4. Create a bundle property for Mobile_Inventory_Inquiry with a value 'Inventory Inquiry' in the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn/ extnbundle.properties` file.
   This creates a mobile device menu option called Inventory Inquiry.
   For more information about creating a custom menu, see Section 3.18, "Customizing the Menu Structure" on page 82.

Choosing this Menu option eventually invokes the JSP defined in the inner panel associated with this entity.

For example, in the rfinventory UI entity, the `/extn/rf/wms/inventory/fmInventory.jsp` file relative to the Sterling Supply Chain Applications base URL is invoked.

5. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

5.5 Creating a Template HTML

A template HTML enables rendering the look and feel of the mobile device screen. A template HTML defines which fields are included and how they are laid out. Each screen requires a template HTML file. The template HTML uses some custom mobile device tags in addition to the standard HTML tags.

The template HTML must adhere to the XSD defined in the `<YANTRA_HOME>/Applications/Foundation/template/mobilescreens/rf.xsd` style sheet.

All template HTML files must reside the `<YANTRA_HOME>/Applications/Foundation/template/extn/mobilescreens/<uientity>` directory. The name <uientity> refers to the screen entity resource that needs to be created for a mobile transaction.

For standard HTML tags of Sterling Supply Chain Applications, see Appendix C, "Console JSP Interface Extensibility Reference".

For standard Sterling Supply Chain Applications mobile UI screen tags, see Appendix D, "Mobile User Interface Extensibility Reference"

Example 5–2   Inventory Inquiry Template HTML

The template HTML files for the inventory inquiry scenario are available for you to copy and reuse from the `<YANTRA_HOME>/Applications/Foundation/documentation/code_examples/rfinventory/` directory.

5.6 Creating JSP Files

The JSP files call the appropriate API (if needed), pick up the appropriate template XML and pass the values returned by the API as data to the template XML.

A separate JSP file must be written for *each* of the following screen components:

- Screen - for invoking the HTML template and rendering the screen on the Sterling Supply Chain Applications mobile client.

- Validation - for performing field level validations.

- Command button - for performing actions on clicking of the command button.

5.6.1 Understanding the Structure of a JSP File

A JSP file for a mobile UI screen typically contains three sections, but not all bullets apply to every mobile JSP file:

Section 1

- Extracts values from the pageContext or Session.

- Performs setAttribute for some of the input bindings.

Section 2

- Calls an API using callAPI.

- Processes logic for the API output.

Section 3

- Sends Form or Forward Page

- Sends Error

Example 5–3   Inventory Inquiry JSP Files

The example JSP files for the inventory inquiry scenario are available for you to copy and reuse from the `<YANTRA_ HOME>/Applications/Foundation/documentation/code_ examples/rfinventory/` directory.

5.6.2 JSP File Name and Directory Guidelines

When naming JSP files for mobile devices, use the following rules:

- The starting JSP file can have any name but the same name must be defined while adding the inner panel for the screen entity.

- The validation JSP file name syntax must be formName + "Val" + fieldname + ".jsp" format (for example, frmSearchValtxtItemId.jsp is invoked for validating the `txtItemId` field on the tab out of the `txtItemId` field in the `frmSearch` form).

- The name of the JSP being called on the click of a button must be named after the value of the action property in the URL. For example, if button has the URL value of `/console/rfinventory.ppc?action=frmSearchUpdCmdInquire`, then the JSP being invoked is named as `frmSearchUpdCmdInquire.jsp`.

- All JSPs must be added to the `<YANTRA_ HOME>/Applications/Foundation/webpages/extn /rf/wms/<uientity>` directory, (for example, <YANTRA_

```
HOME>/Applications/Foundation/webpages
/extn/rf/wms/inventory/frmInventory.jsp).
```

- These JSPs use common utility methods as defined in the `<YANTRA_ HOME>/Applications/Foundation/webpages/yfc/rfutil.jspf` file. For these utility methods, see Appendix C, "Console JSP Interface Extensibility Reference".

5.7 Passing Data Between Screens

Because of the small screen size, the data may need to be persisted from one screen to the next before finally being posted. When passing data between screens, you can either use hidden text or the TempQ utilities. We recommend the usage of TempQ. A TempQ stores the name/value pair information on one page in the session and provides for accessor methods on the subsequent pages. For details on these utilities, see Appendix D, "Mobile User Interface Extensibility Reference".

5.8 Error Handling

Error handling is taken care of by the utility method getError XML as shown in the `rfutil.jspf` file. For information on the getError XML, see Section D.3.4, "deleteFromTempQ" on page 711.

Validations performed by tabbing out of a field or by clicking a button may lead to an error. These errors are displayed in a standard error XML in the response output stream.

```
<errors>
    <error errortxt="errorDesc" focus="errorField"/>
</errors>
```

See the following Inventory Inquiry error message example.

Figure 5–7   Error Messages Displayed

Error Condition 1:
Item ID is not entered

6

Customizing the Sterling Rich Client Platform Interface

This chapter explains how to customize the Sterling Rich Client applications that use the Sterling Rich Client Platform (RCP) User Interface (UI). Additionally, this chapter this chapter provides a brief glimpse into the Sterling Rich Client Platform (RCP) and how to use the Sterling RCP Extensibility Tool to customplug-inize the Sterling Rich Client application UIs is also provided.

6.1 Before You Begin

Before you start customizing the Sterling RCP UI, familiarize with the guidelines provided in this section in order to customize the screens easily and quickly, with fewer errors.

Understanding the Prerequisite Concepts

Before customizing the Sterling RCP UI, it is important that you understand the various concepts of the Sterling RCP. For more information about the various Sterling RCP concepts, see Section 6.2, "Introducing the Sterling Rich Client Platform".

Extensibility Capability Summary

Before you start extending any Sterling Rich Client application UI, it is necessary to understand the extensibility capabilities provided by the Sterling RCP. For more information about extensibility capabilities, see Section 6.3, "Extensibility Capability Summary".

Preparing for Smooth Updates and Easy Maintainability

When you customize any application that uses the Sterling RCP UI, do not modify:

- Plug-in files

- Sterling Supply Chain Applications related resource files

- JAR files

These files are shipped as a part of the standard default configuration. Instead, you can create new files or replicate the files and modify them.

Setting Up the Development Environment

To customize any application, set up the development environment to accommodate modifications that you make to the Sterling Rich Client application UI. For more information about setting up the development environment, see Section 6.4, "Setting Up the Development Environment".

6.2 Introducing the Sterling Rich Client Platform

This section provides a brief glimpse into the Sterling RCP and elaborates on the benefits and concepts of the Sterling RCP.

6.2.1 Overview

The Sterling RCP provides a highly interactive Rich Client, which you can remotely deploy, update, and easily manage. A Rich Client is a client that processes bulk of data operations without depending on the server to which it is connected. However, it is dependent on the server primarily for data storage. Rich in features and functionality, the Rich client has complete access to the programming functions of the operating system.

The Rich Clients are designed such that you can work over a low bandwidth network connections and efficiently utilize the client-side capabilities to avoid costly round trips to the central server. The Rich Clients also provide the ability to work offline.

The Sterling RCP is built on the Eclipse RCP. The Sterling RCP has extended the Eclipse RCP to provide additional features and functionality. In addition to the features provided by the Eclipse RCP, the Sterling RCP provides features such as localizing, theming, binding, and so forth. The

UIs for the Sterling Supply Chain Applications Package Composite Applications (PCAs), such as Sterling Supply Chain Applications Store Operations (SOP) and Sterling Supply Chain Applications Customer Order Management (COM) are developed using the Sterling RCP. Figure 6–1 depicts the Sterling RCP architecture.

Figure 6–1 **Sterling Rich Client Platform Architecture**

6.2.2 Benefits of Sterling Rich Client Platform

The benefits of using the Sterling RCP are:

- Rich User Experience

  - High responsiveness during information retrieval. This does not work in a "page-at-a-time" paradigm of Hyper Text Markup Language (HTML).

  - Ability to validate client side data, if needed.

  - Highly interactive and graphical user interface.

  - Provides visibility to multiple pages on the screen without refreshing any page.

  - Ability to locally store data in memory.

  - Batch server operations.

– Ability to interact with other Desktop applications such as e-mail and spreadsheets.

- Lower Total Cost of Ownership (TCO)

  – Ability to automatically update the Sterling Rich Client applications to remote clients based on the server-side update information.

  – Centralized administration, setup, and client updates.

  – Ability to work across Wide Area Network (WAN) through multiple security infrastructures such as proxies, Firewalls, and so forth.

  – Built-in support for data compression and batch command processing results in optimal network utilization.

  – Ability to work with standard protocols such as Hyper Text Transfer Protocol (HTTP) and Hyper Text Transfer Protocol Secure (HTTPS).

- Deployment Strategy

  – Sterling Rich Client applications are self contained Desktop applications, and depend on the Java Runtime Environments (JREs). The Sterling Rich Client applications can be copied to the Sterling Supply Chain Applications directory and point them to the specific JRE. This provides a reliable and coexisting application deployment strategy without disrupting other existing Java installations. For information about JRE versions that Sterling RCP supports, see the *Sterling Supply Chain Applications Installation Guide*.

  – Subsequent upgrades can be automated with Sterling Supply Chain Applications auto-update feature, which checks and updates the existing configuration on a server. If you do not want to use the auto-update feature, you can turn it off and remotely perform a manual file copy based update.

  – Since basic installation and upgrade involves a "file copy" operation, administration and maintenance can be done locally or remotely.

  – Rich Clients support standard input devices such as keyboard, mouse, stylus, barcode scanner, and so forth. Rich Clients also support standard printers within the supported operating systems.

- Since Sterling Rich Client is a Desktop application, any standard mechanism such as desktop shortcut or program file links can be used to launch the application.

6.2.3 Sterling Rich Client Platform Concepts

The Sterling RCP is built on top of the Eclipse Rich Client Platform (RCP). It extends the functionality provided by the Eclipse RCP and adds on Sterling Supply Chain Applications related functionality to the Eclipse RCP by creating its own editors, views, plug-ins, screens, resource files, and so forth. The Sterling RCP also provides additional features such as localization, theming, binding, creating commands, and so forth. You can use the Sterling RCP to develop Desktop applications.

The Sterling RCP concepts include:

- Sterling Rich Client Applications

- What is XML Binding?

- What is Localization?

- What is Theming?

- What are Related Tasks?

- What are Shared Tasks?

- What is Wizard Definition?

- What is Debug Mode?

- What is Prototype Mode?

- Tracing an Application

- Fetching Images from Server

- Security Handling

- Output Template

- Commands

- Log Files

6.2.3.1 Sterling Rich Client Applications

The Sterling RCP supports Desktop applications. A Desktop application or Multiple Document Interface (MDI) Application contains standard menus,

views, editors, and so forth. You can work with multiple views and editors simultaneously. You can switch from one editor to another without closing any editor that is already open. You can use any standard mechanism (such as desktop shortcut or program file links) to launch the Desktop application. A Desktop application allows you to open one or more documents at the same time and displays each document in a separate window. The menu bar for a Desktop application appears on the application frame. Some examples of Desktop applications include Sterling COM PCA and Sterling SOP PCA. These PCAs are developed using the Sterling RCP.

6.2.3.2 What is XML Binding?

To easily create UIs, the core classes in the Sterling RCP support XML Binding for different types of controls to an XML Distributed Object Model (DOM). This allows the UI developer to bind different controls on a form to various parts of the DOM. The advantage of using XML Binding is that the developer has to write limited code for displaying and/or retrieving data from a specific field in any screen. The developer has to only set and get the model of a screen to set and get the data for the entire screen.

The XML Binding is performed to map the input XML to the screen and back from the screen to an output XML. The XML Binding in the Sterling RCP is XML driven. A binding definition is an XPath (XML Path), which defines the rules for retrieving data from one XML and sending it to another XML. You can set the XML Bindings for various controls such as text boxes, combo boxes, buttons, tables, and so forth. The XML Bindings are specified to associate controls on the screen with a model (an XML document that stores information). To associate the controls to a model, XML Bindings are specified. Usually, it is an XPath specifying the attribute or an element in the document. The XML Binding used depends on the type of control that is used. The Sterling RCP provides various XML binding data classes for different controls. For more information about XML binding classes, see Section 6.8.5.1, "Binding Classes".

6.2.3.3 What is Localization?

The Sterling Rich Client applications are all internationalized. This means they can handle multiple languages and cultural conventions transparently. The Sterling RCP enables you to customize the Sterling Rich Client applications such that the extensions are also

internationalized. The user can localize all graphical text and messages. You can localize the Sterling Rich Client application by defining locale-specific entries in the bundle file. For more information about localizing Sterling Rich Client application, see the *Sterling Supply Chain Applications Localization Guide*.

6.2.3.3.1 Database Localization

In addition to storing the transaction data, the database also stores configuration data, such as error codes and item descriptions of various attributes. This means that the database may need to store values in a language-specific format. If these database literals are not localized, screen literals displays inconsistently, with some displaying in the localized language and others displaying in English. You can store item descriptions in your database in multiple languages. If localizing Sterling Rich Client application UIs, you may want to localize the factory setup. For more information about database localization, see the *Sterling Supply Chain Applications Localization Guide*.

6.2.3.4 What is Theming?

The theming feature enables users to define different fonts, colors used within the applications by creating a custom theme. For more information about theming controls, see Section 6.17, "Theming the Sterling Rich Client Application".

6.2.3.5 What are Related Tasks?

This feature allows you to extend the Sterling Rich Client applications by adding (or hiding) end user tasks in the UI. These tasks are available to the end user through a common related tasks view. Related Tasks feature enables you to perform the tasks that are related to a particular operation. You can group a set of related tasks by associated them with a group. You can also define a category, which can contain multiple tasks from multiple groups. For example, if you are viewing the details of an order, then all the related tasks for this operation such as Cancel Order, Add Order Line, and so forth are displayed in the Related Tasks view under the Order group. You can provide the implementation for displaying these related tasks on the screen. You can also provide the implementation for opening extensible related tasks in a Sterling Supply Chain Applications-provided editor or in your own custom editor. For creating the related tasks you need to extend the following extension points:

- YRCRelatedTasks extension point

- YRCRelatedTaskCategories extension point

- YRCRelatedTaskGroups extension point

- YRCRelatedTasksDisplayer extension point

- YRCRelatedTasksExtensionContributor extension point

For more information about adding new related tasks and hiding existing related tasks, see Section 6.15, "Adding New Related Tasks and Hiding Existing Related Tasks".

6.2.3.6 What are Shared Tasks?

Sterling RCP-based applications such as Sterling COM PCA may contain some reusable UI components such as lookup screens. In such cases, the other Sterling RCP-based applications or extension plug-ins do not have to recreate the same UI components. Instead, they can use the available UI components as a shared task.

To maintain the backward compatibility and avoid multiple plug-in dependencies, the shared tasks are registered with the Sterling RCP plug-in. The other Sterling RCP-based applications or extension plug-ins can directly invoke these shared tasks using the utility methods provided by the Sterling RCP plug-in.

You can register the shared tasks through the YRCSharedTasks extension point defined in the Sterling RCP plug-in. To use these registered shared tasks in your application, invoke them by clicking a button or menu item. For more information about registering and using the shared tasks, see Section 6.16, "Registering and Using the Shared Tasks".

6.2.3.7 What is Wizard Definition?

The wizard definition defines the flow of the wizard. You can define new wizard rules to control the flow of the wizard. The flow of the wizard depends on the output value of a wizard rule. The output of the wizard rule is compared with the transition value. Transition lines are used to transfer control from one wizard entity to another wizard entity.

A Wizard definition consists of following:

1. Wizard Entity—There are two types of wizard entities:

- Wizard Page - The wizard page takes care of the UI to take the inputs from the user.

- Wizard Rule - Wizard rule is a logical step that performs some computations to evaluate different output values. Based on these output values the wizard transitions are defined to decide the flow of the wizard.

2. Wizard Transition—Wizard transition is used to transfer control from one wizard entity to another wizard entity. Wizard transition connects a sequences of wizard entities with each other.

You can start your wizard with a wizard rule or a wizard page. For the wizard entity from where the wizard definition starts, the `start` property should be set to true. You can add transition lines to transfer the control from one wizard entity to another wizard entity based on the output values of the wizard rule. The transitions originating from a wizard page can have only one target. Transitions starting from a wizard rule can have multiple targets based on the rule output. The output of a wizard rule is compared with the transition values defined for a rule. Based on this value the control is transferred to the appropriate wizard entity.

All the new wizard definitions are created in the `<Plug-in_id>_<wizard_ name>.ycml` file.

> **Note:** You must use a separate `<Plug-in_id>_<wizard_ name>.ycml` file for each wizard definition.

Let us consider that you have a the wizard definition for the following wizard flow:
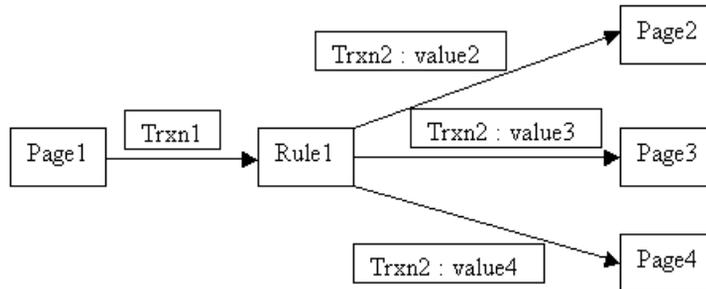
The wizard starts from a wizard page (Page1) and transitions to a wizard rule (Rule1) based on a value. From the wizard rule (Rule1) the wizard ends at three different wizard pages based on three different values. For the wizard entity from where the wizard definition starts, the `start` property should be set to true.

- The wizard page (Page1) contains the wizard page identifier (id) and the implementation class (impl).

    > **Note:** A wizard page cannot have multiple transitions. You can only have only one transition starting from a wizard page and ending at another wizard entity.

- The wizard rule (Rule1) contains the wizard rule identifier (id), implementation class (impl), namespace definition for the rule (Namespace), and the list of output values one of which is the output of the wizard rule.

    > **Note:** Multiple transitions can take place from a wizard rule. Therefore, a wizard rule can return multiple output values, one value for each transition that starts from the wizard rule.

- There are two types of transitions:
    - Transition from a wizard page—There is a transition from a wizard page (Page1) and ends at wizard rule (Rule1). The wizard

transition that starts from a wizard page contains wizard transition identifier (id), source wizard entity (source) and the target wizard entity (target). The source contains the identifier of the wizard entity from where the transition starts and target contains the identifier of the wizard entity at which the transition ends.

— Transition from a wizard rule—There are three transitions that are starting from a wizard rule (Rule1). These three transitions end at three different wizard pages such as "Page2", "Page3", and "Page4" based on the output value returned by the wizard rule (Rule1).

Note: As a wizard rule can have multiple transitions. Therefore, when a transition starts from a wizard rule you can define multiple targets with the corresponding value associated with each target that you define.

The wizard transition that starts from a wizard rule contains wizard transition identifier (id), source wizard entity (source), multiple target wizard entities (target), and the output value of the wizard rule for which a transition takes place. The source contains the identifier of the wizard entity from where the transition starts and target contains the identifier of the wizard entity at which the transition ends.

Note: The transition identifier (id) for all the transitions that start from a wizard rule are same. The transition takes place based on the value defined for a given transition.

6.2.3.8 What is Debug Mode?

Sterling RCP allows you to run a Sterling Rich Client application in debug mode. While running in debug mode, the Sterling RCP performs some additional validations on the Sterling Rich Client application to reduce the number of errors created while making extensions. Also, when you run an application in debug mode, the Sterling RCP provides additional visual information about errors and missing parameters.

> **Note:** If the control name is not specified for a control, then the background of that particular control on the screen is painted red when the application is run in debug mode.

The Sterling RCP performs following validations in debug mode:

1. Control Name Validation—The Sterling RCP checks whether or not a unique control name has been specified for each control in the Sterling Rich Client application. If this is not specified, when you move the cursor on that control, the tool tip displays "Specify the name of the control". Additionally, the control is displayed with a red background. This is useful in extending a Sterling Rich Client application. You can easily extend a screen if you know the name of all controls on the screen.

2. Bundle Entry Validation—The Sterling RCP checks whether you have specified the bundle entry in the `*bundle.properties` file for each string in the Sterling Rich Client application. If you do not specify the bundle entry for a string, it is considered as a non-localized string. Such non-localized strings always displays within the exclamation marks (!). For example, if you do not specify the bundle entry for the "Order No" string, the string display as !Order No!. This bundle entry validation helps the developers to understand whether the strings on the UI are localized or non-localized.

> **Note:** The localized strings are sometimes displayed within the exclamation marks.

6.2.3.8.1 Running the Sterling Rich Client Application in Debug Mode

You can run the Sterling Rich Client application in debug mode in order to perform some extra validations and traces. Additionally, debug mode provides much more visual information to tell you what is wrong and where. You can run both a standalone application and an application within Eclipse in the debug mode. This section explains the following:

Running the Standalone Sterling Rich Client Application in Debug Mode

You can run the standalone Sterling Rich Client application in debug mode. The standalone application can be a shipped application or an extended application.

To run the standalone Sterling Rich Client application in debug mode:

1. Modify the Sterling Rich Client application's `*.ini` file to provide the appropriate VM arguments to run the application in debug mode. You can find the `*.ini` file for the Sterling Rich Client application in the `<YANTRA_HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/<PCA_DIR>/` directory.

   where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications..

   For example, if you want to run the Sterling COM PCA in debug mode, edit the `<YANTRA_HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/com20/com20.ini` file.

2. In the `*.ini` file add the following VM arguments:

   ```
   -vmargs
   -Ddebugmode=true
   ```

3. Run the EXE file of the Sterling Rich Client application.

Running the Sterling Rich Client Application within Eclipse in Debug Mode

While launching the Sterling Rich Client application in Eclipse, in the VM Arguments field, enter the following arguments:

   ```
   -Ddebugmode=true
   ```

For more information about launching the Sterling Rich Client application in Eclipse, see Section 6.4.5, "Launching the Sterling Rich Client Application in Eclipse".

6.2.3.9 What is Prototype Mode?

Sterling RCP allows you to run a Sterling Rich Client application in prototype mode. This is useful as it provides the ability to run a Sterling Rich Client application without the need to connect it to a server. Also, when you run a application in prototype mode, the calls to the APIs or

services do not go to the server, instead they use the sample output XML files. You must store these sample output XML files in the `prototype` directory. The output from these sample output XML files is hard-coded and is not meant to reflect any authentic data. This is merely to testing or verifying the UI that you develop. Prototype mode is useful when you want to test the UI as you develop and you do not want to go to the server for API or service outputs.

6.2.3.9.1 Running the Sterling Rich Client Application in Prototype Mode

You can run the Sterling Rich Client application in the prototype mode to test UIs. The Sterling RCP enables you to run a standalone application and applications within Eclipse in the prototype mode.

Running the Standalone Sterling Rich Client Application in Prototype Mode

You can run the standalone Sterling Rich Client application in prototype mode. The standalone application can be any application that is already shipped or an extended application.

To run the standalone Sterling Rich Client application in prototype mode:

1. Modify the Sterling Rich Client application's `*.ini` file stored in the `<YANTRA_HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/<PCA_DIR>/` directory to provide appropriate VM arguments.

   where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

   For example, if you want to run the Sterling COM PCA in prototype mode, modify the `<YANTRA_HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/com20/com20.ini` file.

2. In the `*.ini` file add the following VM arguments:

   ```
   -vmargs
   -DProtoTypeDir=C:/EclipseInfrastructure/com.yantra.yfc.rcp.ri/prototype
   ```

   where `ProtoTypeDir` property refers to the `prototype` directory that contains the sample output XML files.

3. Make sure that the name of all sample output XML files stored in the `prototype` directory are same as the command name for which they are used. For example, if the command name is getOrderDetails, the

sample output XML file used for this command must be named as
`getOrderDetails.xml`.

4. To run a command in prototype mode, in the commands file, set the
value of the `prototype` attribute for that particular command to
"true". For more information about creating commands, see
Section 6.13, "Creating Commands".

> **Note:** The prototype mode is always set at the command
> level. It is important that you set the value of the
> `prototype` attribute to "true". This invokes the API or
> service in prototype mode.

5. Run the EXE file of the appropriate Sterling Rich Client application.

**Running the Sterling Rich Client Application within Eclipse in
Prototype Mode**
While launching the Sterling Rich Client application in Eclipse, in the VM
Arguments field, enter the following argument:

`-DProtoTypeDir=C:/EclipseInfrastructure/com.yantra.yfc.rcp.ri/prototype`

where `ProtoTypeDir` property refers to the `prototype` directory that
contains the sample output XML files.

> **Note:** Make sure that the name of all sample output XML
> files stored in the `prototype` directory are same as the
> command name for which they are used. For example, if
> the command name is getOrderDetails, the sample output
> XML file used for this command must be named as
> `getOrderDetails.xml`.

For more information about launching the Sterling Rich Client application
in Eclipse, see Section 6.4.5, "Launching the Sterling Rich Client
Application in Eclipse".

6.2.3.10 Tracing an Application
Sterling RCP enables you to trace a specific Sterling Rich Client
application. This is useful in checking the operations such API or service

calls, informations, warning, errors (if any), bindings, and so forth. When you start tracing an application, the system writes all information in the log file. You can trace both standalone application and any application within Eclipse. This section explains the following:

Tracing the Standalone Sterling Rich Client Application

You can set the trace on for a standalone Sterling Rich Client application. The standalone application can be any application that is shipped or an extended application.

To start tracing a standalone Sterling Rich Client application:

1. Modify the Sterling Rich Client application's `*.ini` file stored in the `<YANTRA_HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/<PCA_DIR>/` directory to provide appropriate VM arguments.

   where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

   For example, if you want to trace the Sterling COM PCA, edit the `<YANTRA_HOME>/Applications/rcpdrop/<OPERATING_ SYSTEM>/com20/com20.ini` file.

2. In the `*.ini` file add the following VM arguments:

   ```
   -vmargs
   -Ddebugfile=C:/debug.log
   ```

   where `debugfile` property refers to the directory in which the log file is created.

3. Run the EXE file of the appropriate Sterling Rich Client application.

4. After you successfully log in to the application, the application window displays. To start or stop tracing the application, press CTRL+F2.

> **Note:** You can turn the trace ON for an application by setting the `trace` property to "true" in the `*.ini` file. For example, set the value of -Dtrace=true. When you start the application, by default, the trace is set ON. To stop tracing the application, press Ctrl+F2.

Tracing the Sterling Rich Client Application within Eclipse

While launching the Sterling Rich Client application in Eclipse, in the VM Arguments field, enter the following argument:

```
-Ddebugfile=C:/debug.log
```

where `debugfile` property refers to the directory in which the log file is created.

For more information about launching the Sterling Rich Client application in Eclipse, see Section 6.4.5, "Launching the Sterling Rich Client Application in Eclipse".

6.2.3.11 Fetching Images from Server

The Sterling RCP allows you to fetch images from the server on labels and table columns. To fetch images from the server, define a Config element in the `location.ycfg` file and set the connection settings for fetching images. You can define multiple Config elements to fetch images of different formats. The Sterling RCP supports various image formats such as GIF, BMP, ICO, JPEG, PNG, and TIFF. For information about configuring the connection settings for fetching images from the server, see the *Sterling Supply Chain Applications Installation Guide*.

> **Note:** You can fetch images from the server only for labels and table columns.

6.2.3.12 Security Handling

Sterling RCP enables you to securely communicate with the servers. The Sterling RCP allows you to connect to servers using the HTTPS protocol. This provides an authenticated and encrypted way of running the resources from the server on the client machine. The authentication and encryption is handled using certificates. These certificates helps you run

the authorized resources on client machines. These certificates must be stored in the `truststore` folder of the Sterling RCP plug-in.

To connect to the server using the HTTPS protocol, specify the protocol as HTTPS and also specify the port number for the HTTPS connection in the configuration file. For more information about configuring the connection settings for HTTPS connection, see the *Sterling Supply Chain Applications Installation Guide*.

Selective SSL Calls

For a HTTP connection, you can make selective SSL calls to certain sensitive APIs at the following levels:

*   API level—You can specify the name of the API for which you want to always make an SSL call. For example, in an application you may always want to call a specific API such as getCreditCardDetails API using an SSL call. In such situations, you can specify these APIs at the API level.

*   Command Level—You can specify the names of the command that you want to call using an SSL. You can filter such commands based on the attributes of an API. For example, in an application, if the changeOrder API is called more than 40 times in different screens, but in two screens this API is called to get or pass the credit card number using the `CreditCardNo` attribute. Now, only for these two screens you want the changeOrder API to be called using an SSL. To know the commands that use the changeOrder API with the `CreditCardNo` attribute, search for the commands and specify the names of such commands at the command level.

You must define the names of all APIs and commands that you want to call securely in the `secureapis.xml` file. You can add these secure APIs to the `secureapis.xml` file using the Sterling RCP Extensibility Tool. For more information about adding the secure APIs using the Sterling RCP Extensibility Tool, see Section 6.20.1.10, "Adding Secure APIs".

To configure selective SSL calls for APIs, in the `locations.ycfg` file, specify the `HttpsPortNumber` additional attribute in the Config element. For more information about configuration settings for making selective SSL calls for some APIs, see the *Sterling Supply Chain Applications Installation Guide*.

6.2.3.13 Output Template

Output templates are used during an API call or service to ensure that the data is retrieved in the desired format. For example, if you want to display few attributes in a particular order.

You can also merge output templates to retrieve the additional data from an API or service. For more information about template merging, see Section 6.14, "Template Merging".

6.2.3.14 Commands

Sterling RCP has modelled API calls as commands. Commands are defined to call APIs or services to retrieve data in the desired format. Whenever you call an API, specify the name of the command associated with the API. Sterling RCP supports creating commands at a form level. You can also override commands, if necessary. This is useful when you want to call a custom API for a particular form. For more information about creating commands, see Section 6.13, "Creating Commands".

6.2.3.15 Log Files

Log files contain information such as warnings and errors (if any). When you run a Sterling Rich Client application, the following log files get generated:

*   Eclipse log file—Whenever you run a Sterling Rich Client application, the Eclipse automatically creates a log file. This log file contains high-level error and/or warnings logged by Eclipse, and is always be created.

    Using the `osgi.instance.area.default` property, you can configure the location to store this log file in the `<YANTRA_ HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/<PCA_ DIR>/configuration/config.ini` file. By default, this log file is stored in the `@user.home/<application_workspace>` directory. Sterling RCP does not allow you to rename the log file.

    For example, if you run the Sterling COM PCA, the `config.ini` file is stored in the `<YANTRA_HOME>/Applications/rcpdrop/<OPERATING_ SYSTEM>/COM/configuration` directory.

    where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

In the `config.ini` file, the `osgi.instance.area.default` property is set to:

`osgi.instance.area.default=@user.home/com20workspace`

where `@user.home` refers to the user's home directory. You can only change the directory where the log file gets created, but cannot change the name of the file.

- RCP Infrastructure log file—Whenever you run a Sterling Rich Client application for which the tracing is turned ON, Sterling RCP plug-in creates a log file. This log file provides information about API or service calls, warning or error messages (if any), bindings, and so forth. You can specify the location in the `*.ini` file of the appropriate application, which you want to trace. The `*.ini` file is located in the `<YANTRA_HOME>/Runtime/rcpdrop/<OPERATING_SYSTEM>/<PCA_DIR>/` directory.

  where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

  > **Note:** Make sure that the tracing is turned ON in the application for the information to be written in the log file.

  For more information about tracing an application, see Section 6.2.3.10, "Tracing an Application".

6.3 Extensibility Capability Summary

The Sterling RCP provides various extension points, which you can implement to extend the Sterling Rich Client application as needed. Using features such as Localization and Theming, you can further extend the Sterling Rich Client application. The Sterling RCP also provides an Sterling RCP Extensibility tool using which you can extend the Sterling Rich Client applications UI.

You can extend the Rich Client application by:

- Adding or Removing Menus—You can add or remove menus from the Sterling Rich Client screens by defining a new resource in the resources of Sterling Supply Chain Applications. For more information

about adding or removing menus, see Section 6.18, "Adding or Removing Menus".

- Creating and Adding New Screens—You can create new Sterling Rich Client screens for a Sterling Rich Client application. You can also add the newly created Rich Client screens to a Sterling Rich Client application. For more information about creating and adding new screens, see Section 6.5.2.3, "Creating and Adding New Screens".

- Adding New Related Tasks and Hiding Existing Related Tasks—You can add new related tasks and hide existing related tasks from the Sterling Rich Client applications. For more information about adding and hiding related tasks, see Section 6.12, "Creating Related Tasks".

- Modifying Existing Screens—You can modify existing screens of a Rich Client application using the Sterling RCP Extensibility Tool. For information about modifying existing screens, see Section 6.6, "Modifying Existing Screens".

- Modifying Existing Wizards—You can modify existing wizards of a Rich Client application using the Sterling RCP Extensibility tool. For more information about modifying existing wizards, see Section 6.5.2.2, "Modifying Existing Wizards".

- Localizing—You can localize the Rich Client application for different languages based on the user's locale. The user can localize the Sterling Rich Client application by defining locale-specific entries and translating the text. For more information about localizing the Sterling Rich Client application, see Section 6.5, "Customizing the Sterling Rich Client Application".

- Theming—You can customize the Rich Client application by using custom theme. You can change the font type and color scheme for controls, graphical text, messages, and so forth. For more information about theming, see Section 6.17, "Theming the Sterling Rich Client Application".

6.4 Setting Up the Development Environment

Before you start customizing the Sterling Rich Client UI, you must set up the development environment to perform customizations. This involves:

- Installing Prerequisite Software Components
- Creating and Configuring Locations

- Creating a Plug-in Project

- Running the Sterling RCP Plugin Wizard

- Launching the Sterling Rich Client Application in Eclipse

6.4.1 Installing Prerequisite Software Components

This section describes the various software components required to customize the Sterling Rich Client application. Before deploying the Sterling Rich Client application, make sure that you have already installed the following software components:

- Eclipse SDK

  Install the Eclipse SDK version that Sterling RCP supports. For more information about the Eclipse SDK version, see the *Sterling Supply Chain Applications Installation Guide*.

- Eclipse Related Plug-ins

  Install the following Eclipse related plug-ins that Sterling RCP supports. For more information about the Eclipse plug-in versions, see the *Sterling Supply Chain Applications Installation Guide*.

  − VE (Visual Editor) plug-in

  − GEF (Graphical Editor Framework) plug-in

  − EMF (Eclipse Modelling Framework) plug-in

- Java Development Kit (JDK)

  Install the JDK version that Sterling RCP supports. For more information about the JDK version, see the *Sterling Supply Chain Applications Installation Guide*.

- Sterling RCP plug-ins

  Install the following Sterling RCP plug-ins that Sterling RCP supports. For more information about the Sterling RCP plug-ins version, see the *Sterling Supply Chain Applications Installation Guide*.

  − Sterling RCP plug-in

  − Sterling RCP Tools plug-in

  These plug-ins are shipped along with Sterling Supply Chain Applications, and are located in the following directory:

`<YANTRA_HOME>/Applications/Foundation/rcpclient`

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

Note: If you are installing a new version of the Sterling RCP plug-ins or updating the previous versions you must clean the cached build information in Eclipse.

To clean this information, start Eclipse with the "-clean" option:

1. Right-click on the Eclipse's shortcut and select Properties from the pop-menu. The Properties window displays.
2. In Target, enter the command line argument "-clean" at the end. For example, `"C:\Eclipse 3.2\eclipse\eclipse.exe" -clean`.
3. Start Eclipse.

Installing the Sterling RCP plug-in

To install the Sterling RCP plug-in:

Extract the `<YANTRA_ HOME>/Applications/Foundation/rcpclient/com.yantra.yfc.rcp_ <version>.zip` file to the `<ECLIPSE_HOME>/plugins` folder.

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications. <ECLIPSE_HOME> refers to the Eclipse SDK installation directory.

Installing the Sterling RCP Tools plug-in

To install the Sterling RCP Tools plug-in:

Extract the `<YANTRA_ HOME>/Applications/Foundation/rcpclient/com.yantra.yfc.rcp.too ls_<version>.zip` to the `<ECLIPSE_HOME>/plugins` folder.

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications. <ECLIPSE_HOME> refers to the Eclipse SDK installation directory.

6.4.1.1 Sterling RCP Tools

The Sterling RCP Tools plug-in contains the following tools:

- Sterling RCP Command XML Editor—The Sterling RCP Command XML Editor provides a way to conveniently edit the `<Plug-in_id>_ commands.ycml` file. The Commands XML file is used to create or modify commands and namespaces.

- Sterling RCP Config XML Editor—The Sterling RCP Config XML Editor tool provides a way to conveniently edit the `locations.ycfg` file. The `locations.ycfg` file contains configuration information for the Sterling Rich Client applications. A location configuration and server configuration must be defined to connect the Sterling Rich Client application to the server.

- Sterling RCP Theme Editor—The Sterling RCP Theme Editor tool provides a convenient way to edit the `<Plug-in_id>_<theme name>.ythm` file. The theme file is used to define theme entries for a particular theme.

- Sterling RCP Wizard Editor—Sterling RCP Wizard Editor tool is used to conveniently edit the `<Plug-in_id>_commands.ycml` for creating or modifying the wizard definition. The wizard definition specifies the flow of a wizard.

> **Note:** To understand how to use the Sterling RCP tool such as the Sterling RCP Command XML Editor, Sterling RCP Config XML Editor, and so forth, in Eclipse, see the cheatsheets provided by the Sterling RCP.
>
> To view the cheatsheets in Eclipse, follow these steps:
>
> 1. Start Eclipse SDK.
> 2. From the menu bar select Help > Cheat Sheets. The Cheat Sheet Selection window displays. Expand "Sterling RCP: UI Editor" from the list and open the appropriate cheatsheet.

- Sterling RCP UI Wizards—The Sterling RCP UI Editor is a plug-in that includes several development time database utilities for Sterling Rich Client application. The Sterling RCP UI Editor provides various wizards for creating these utilities, such as:

- **–** Sterling RCP Composite—Sterling RCP Composite wizard is used to create the standalone Rich Client screens for the Sterling Rich Client application.

- **–** Sterling RCP Plugin—Sterling RCP Plugin wizard is used to extend an Eclipse plug-in so that it can be recognized by the Sterling RCP. The Sterling RCP Plugin wizard includes a plugin file and various Sterling Supply Chain Applications-specific resource files such as the theme file, configuration file, command file, bundle file, and extension file. The plugin java file is used to register the Eclipse plug-in and the Sterling RCP-specific resource files with the Sterling RCP. Whenever you run the Sterling RCP Plugin wizard on top of an Eclipse plug-in, the newly created bundle activator gets updated. Also, the bundle activator for the plugin file is placed in the `plugin.xml` file.

- **–** Sterling RCP Search List Composite—Sterling RCP Search List Composite wizard is used to create the sample search and list screen for the Sterling Rich Client application. The Sterling RCP Search and List screens are divided into search screen and list screen. The Search screen accepts the search criteria entered by the user and performs the search operation. The List screen displays the results of the search operation.

  > **Note:**   To open the UI Wizards:
  >
  > 1. Launch the Sterling Rich Client application in Eclipse. For more information about launching the Sterling Rich Client application in Eclipse, see Section 6.4.5, "Launching the Sterling Rich Client Application in Eclipse".
  > 2. Press Ctrl+N. Expand "Sterling RCP Wizards" from the list of wizards and open the appropriate wizard.

- **–** Sterling RCP Extensibility Tool—The Sterling RCP Extensibility Tool is used to modify the existing screens of a Sterling Rich Client application. Using this tool, you can add new controls, modifying existing controls, and so forth.

- Sterling Rich Client Application Plug-in—Unzip the Sterling Rich Client application's zip file that you want to customize to any directory. You can find the zip file for a PCA in the following directory:

`<YANTRA_HOME>/Applications/<PCA_DIR>/rcpclient/`

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications. `<PCA_NAME>` refers to the PCA installation directory.

For example, if you want to customize the Sterling COM PCA, unzip the `<YANTRA_HOME>/Applications/COM/rcpclient/com20.zip` file to any directory.

After you extract all files, copy the content of the Sterling Rich Client Application's plug-in folder to the `<ECLIPSE_HOME>/plugins` folder.

For example, if you want to customize the Sterling COM PCA, copy the: `<TEMP_UNZIP_DIR>/plugins/com.yantra.pca.ycd.rcp_<version>` to the `<ECLIPSE_HOME>/plugins` folder.

where `<TEMP_UNZIP_DIR>` is the name of the directory where you have extracted the `com20.zip` files. `<ECLIPSE_HOME>` refers to the directory where you have installed Eclipse SDK.

6.4.2 Creating and Configuring Locations

This section explains how to create the `locations.ycfg` XML file for configuring locations.

To configure locations, you must first create the `locations.ycfg` XML file.

To configure a location, follow these steps:

1. In the `locations.ycfg` XML file, define a Locations root element.

2. Under the Locations root element, define the Location element. In the `id` attribute of the Location element, specify the location identifier such as DEFAULT, LOCAL, REMOTE, and so forth.

3. Configure the proxy server and application server URL settings for the location. For more information about location configuration settings, see the *Sterling Supply Chain Applications Installation Guide*.

> **Note:** You must have one Location element with `id` attribute value as "DEFAULT" and this Location element must have Config element whose `Name` attribute should have the value as "DEFAULT".
>
> When you log in to a Sterling Rich Client application using a particular location, the system checks whether or not loaded location has a "DEFAULT" Config element defined for it. If the selected location has "DEFAULT" Config element, the system loads the "DEFAULT" configuration. Otherwise the system loads the "DEFAULT" configuration defined in the "DEFAULT" location.

4. Add `locations.ycfg` XML file to `resources.jar` file.

5. Copy the `resources.jar` file to the `<ECLIPSE_HOME>/plugins/com.yantra.yfc.rcp_<version>` directory

   where `<ECLIPSE_HOME>` refers to the Eclipse SDK installation directory.

6.4.3 Creating a Plug-in Project

This section explains how to create a plug-in project.

To create a plug-in project:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Open Perspective > Other.… The Select Perspective window displays.
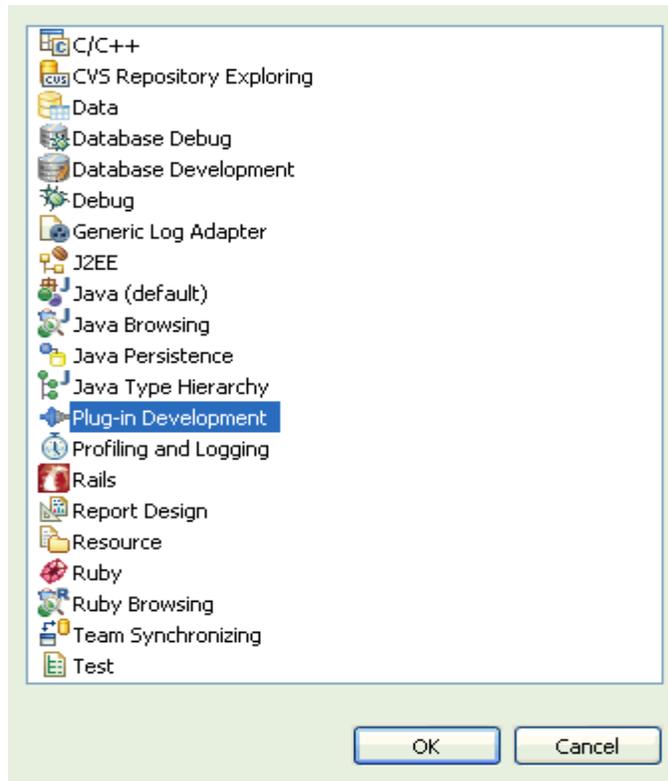
3. From the list of wizards, select Plug-in Development.

4. Click OK. The Eclipse Workbench opens in Java perspective. For more information about the Eclipse workbench, see Appendix B.3, "Workbench".

5. From the menu bar, select File > New > Project…. The New Project window displays.

6. From the list of wizards, under Plug-in Development category, select the Plug-in Project.

7. Click Next. The New Plug-in Project window displays.

Table 6–1   New Plug-in Project Window

Field	Description
Project name:	Enter the name of the new plug-in project.
Use default location	Uncheck this box if you want to specify the path where you want to store the new plug-in project. By default, this box is always checked. Sterling Commerce recommends that you use the default directory to store the new plug-in project.
Project Settings	
Make sure that the Source folder: and Output folder: text boxes are empty.	

Table 6–1   New Plug-in Project Window

Field	Description
Target Platform	
Eclipse version:	Select 3.2 from the drop-down list.

8. Click Next. The Plug-in Content page displays.

9. Click Next. The Templates page displays.

10. Click Finish. The new plug-in project gets created.

6.4.4 Running the Sterling RCP Plugin Wizard

After creating the new plug-in project, run the Sterling RCP Plugin wizard on top of the new plug-in project that you created. The Sterling RCP plugin wizard enables you to extend an Eclipse plug-in so that it is easily understood by the Sterling RCP. The Sterling RCP plugin wizard creates a plugin java file and following Sterling RCP-specific resource files:

- Bundle file such as `*bundle.properties`—used to define bundle entries for internationalizing a Sterling Rich Client application.

- Commands file such as `*commands.ycml`—used to define commands for calling APIs or services to get the required data.

- Theme file such as `*theme.ythm`—used to define theme entries for theming a Sterling Rich Client application.

- Extension files such as `*extn.yuix`—used to store all extensions made to a Sterling Rich Client application.

These resource files allow you to extend the UI and control the behavior of a Sterling Rich Client application. The plugin java file is used to register the Eclipse plug-in and the Sterling RCP-specific resource files with the Sterling RCP.

To run the Sterling RCP Plugin wizard

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displays in the Navigator view.

3. Expand the plug-in project that you created. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. Right-click on the folder where you want to store the Sterling RCP extension plugin java file and select New > Other from the pop-up menu. The New window displays.

5. From the list of wizards, select Sterling RCP Wizards > Sterling RCP Plugin.

6. Click Next. The New Plugin to Sterling RCP UI window displays.

Table 6–2   New Plugin to Sterling RCP UI Window

Field	Description
Source Folder:	The folder path that you selected automatically displays. Click Browse to browse to the source folder where you want to store the plugin java file, if necessary.
Plugin Id	The identifier of the plug-in project which contains the source folder automatically displays.
Package:	Enter the name of the package in which you want to store the plugin java file, if necessary. This helps you to easily manage the directory structure of your plug-in project. If not specified, the system automatically creates a default package.

Table 6–2   New Plugin to Sterling RCP UI Window

Field	Description
Plugin File Name	By default, the NewPlugin.java plugin file name displays. Enter a new plugin file name, if necessary. This plugin file registers your resource files such as bundles, themes, commands, and extension files.
Application File Name	Enter the name of an application file name, if necessary.

7. Click Finish.

After you run the Sterling RCP Plugin wizard on top of a plug-in project, the Sterling RCP performs the following tasks:

• Loads the dependent plug-ins. The dependent plug-ins are the plug-ins whose extension points are extended by another plug-in to extend the functionality provided by the Eclipse platform.

• Implements the YRCPluginAutoLoader extension point. The YRCPluginAutoLoader extension point is provided by the Sterling RCP, which the order in which plug-ins need to be loaded. The YRCPluginAutoLoader extension point automatically loads the classes within a plug-in during startup in the specified order. All classes that need to be automatically loaded are sorted in the ascending order and loaded one at a time. For more information about the YRCPluginAutoLoader extension point, see Appendix B.5, "YRCPluginAutoLoader Extension Point".

• Creates the plugin java file. The plugin java file is stored in the folder that you specified. This file is used to register the plug-in that you created and the Sterling RCP-specific resource files.

• Creates the following Sterling RCP-specific resource files:

    — Bundle file such as *bundle.properties

    — Commands file such as *commands.ycml

    — Theme file such as *theme.ythm

    — Extension file such as *extn.yuix extension

Figure 6–2 illustrates a typical folder structure that has both plugin java file and Sterling RCP-specific resource files stored under the package that you specified.

Figure 6–2   Typical Folder Structure in Eclipse
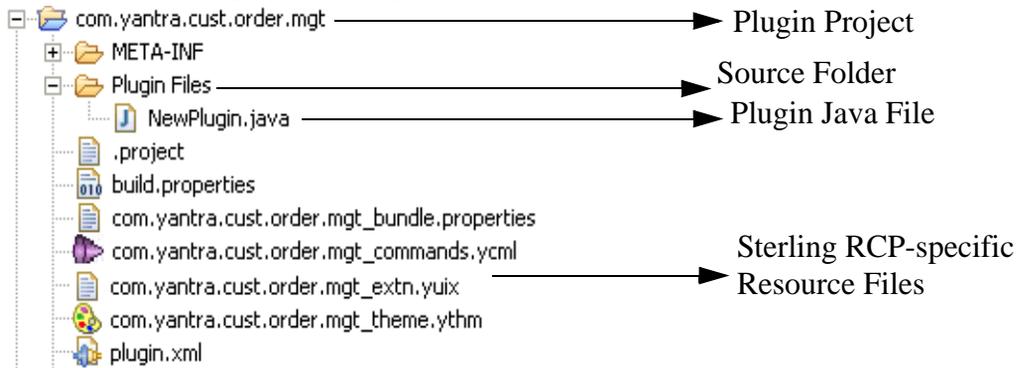
6.4.5 Launching the Sterling Rich Client Application in Eclipse

After you run the Sterling RCP Plugin wizard on the plug-in project, launch the Sterling Rich Client application that you want to customize within Eclipse.

To launch the Sterling Rich Client application with Eclipse:

1. Start Eclipse SDK. In the Package Explorer view, you can see the plug-in project that you created.

2. From the menu bar, select Run -> Run… The Run window displays.
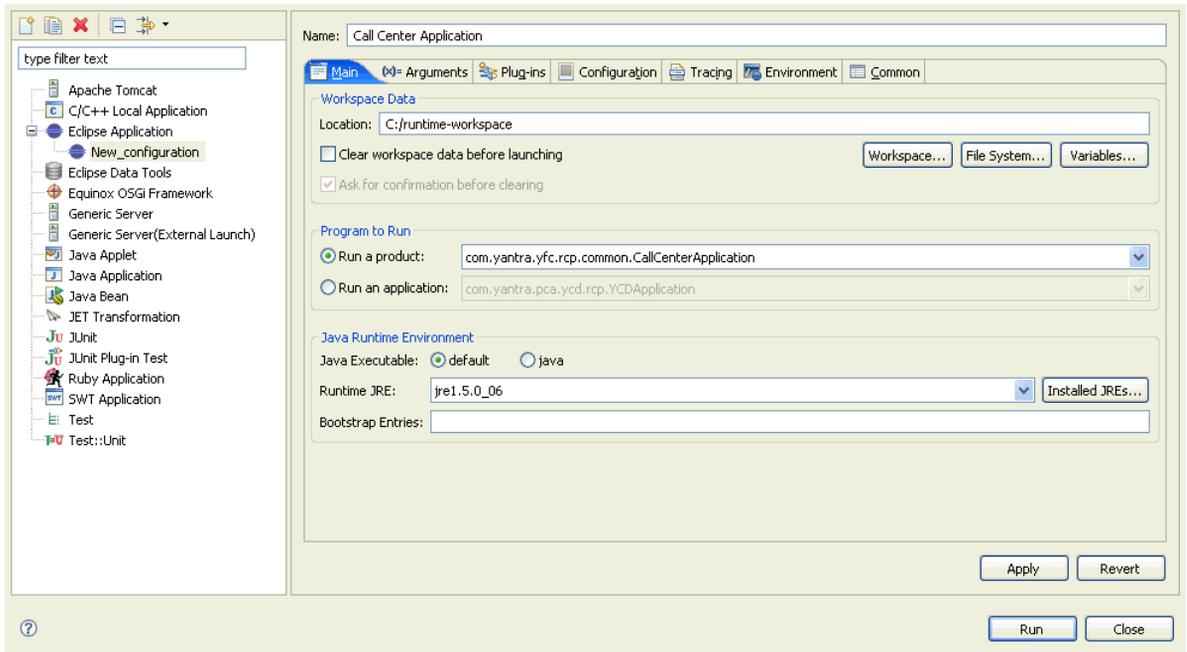
3. In the left panel, right-click and select New_configuration from the pop-up menu.

4. In Name:, enter the name for the new run configuration. For example, Call Center Application.

5. In the Workspace Data panel, in Location:, enter the location where you want to create the runtime workspace. Click File System… to browse to the directory where you want to create the runtime workspace.

6. In the Program to Run panel, choose Run a product: and from the drop-down list select the product identifier of the application you want to customize. For example, if you want to customize the Sterling COM PCA, select the product identifier of the Sterling COM PCA. For more information about the product identifier for each PCA, see the respective PCA Installation Guide.

7. In the Plug-ins tab, choose Choose plug-ins and fragments to launch from the list.

8. Click Deselect All.

9. From the list of plug-ins, expand Workspace Plug-ins and select the plug-in project that you created.

10. Expand External Plug-ins. Select the Sterling Rich Client application plug-in that you want to customize.

> **Note:** Make sure that you select plug-in that is same as the Sterling Rich Client application whose ID you selected in the Step 6.

11. Select the com.yantra.ide.rcptools.rcpextn plug-in.

12. Click Add Required Plug-ins.

13. Click Validate plug-in Set. If you have correctly performed all steps, the Plug-in Validation window displays the message "No problems were detected in the selected set of plug-ins".

14. Click OK.

15. Select Configuration tab and check the "Clear the configuration area before launching" box. This clears the cached configuration data saved by Eclipse.

16. Click Apply.

17. Click Run. The Sterling Rich Client application that you selected in Step 6 now runs.

> **Note:** The Sterling RCP Extensibility tool plug-in depends on some of the Eclipse plug-ins. When you add the Sterling RCP extensibility tool plug-in, these dependent Eclipse plug-ins are automatically added. Therefore, when you launch a Sterling Rich Client application such as Sterling COM PCA within Eclipse, the system throws the following error messages:
>
> Invalid Menu Extension (Path is invalid): org.eclipse.ui.actions.showKeyAssistHandler.
>
> Invalid Menu Extension (Path is invalid): org.eclipse.update.ui.updateMenu.
>
> Invalid Menu Extension (Path is invalid): org.eclipse.update.ui.configManager.
>
> Invalid Menu Extension (Path is invalid): org.eclipse.update.ui.newUpdates.
>
> These are known issues and have no bearing on the functioning of an application.

6.5 Customizing the Sterling Rich Client Application

Sterling RCP supports various ways of customizing a Sterling Rich Client application. You can customize a Sterling Rich Client application in one of the following ways:

- Localization and Theming Sterling Rich Client Application

- Extending the Sterling Rich Client Application

> **Note:** When customizing the Sterling Rich Client application, copy the standard Sterling RCP-specific resource files and modify it or create new resource files. Do not modify the Sterling Supply Chain Applications-specific resource files that are shipped with the Sterling Supply Chain Applications.

6.5.1 Localization and Theming Sterling Rich Client Application

This section explains how to localize and theme a Sterling Rich Client application.

6.5.1.1 Localizing a Sterling Rich Client Application

You can localize the Sterling Rich Client application's locale-specific files based on the user's locale. Sterling RCP supports bundle and theme locale-specific files. The Sterling Rich Client application plug-ins contain bundle file such as `<Plug-in_id>_<name>.properties` and theme file such as `<Plug-in_ id>_<theme_name>.ythm`. For more informationabout localizing bundle and theme files, see the *Sterling Supply Chain Applications Localization Guide*.

6.5.1.2 Theming a Sterling Rich Client Application

You can theme the Sterling Rich Client application based on the custom theme. To theme your application, at the plug-in level, define new theme entries for controls, text, strings, images, and so forth in the `<Plug-in_ id>_<theme name>.ythm` file. For more information about theming the Sterling Rich Client application, see Section 6.17, "Theming the Sterling Rich Client Application".

6.5.2 Extending the Sterling Rich Client Application

You can extend the Sterling Rich Client application's UI to address specific needs of your business. Extending the Sterling Rich Client application can be as simple as defining some additional fields or as advanced as defining an entire new plug-in.

> **Note:**   Sterling Commerce recommends that you extend the Sterling Rich Client application by modifying existing screens.

> **Note:**   Before you can start extending the Sterling Rich Client application using any one of the given ways, make sure that you set up the development environment for performing customizations. For more information about setting up development environment, see Section 6.4, "Setting Up the Development Environment".

You can extend the Sterling Rich Client application by:

- Modifying Existing Screens
- Modifying Existing Wizards
- Creating and Adding New Screens

6.5.2.1 Modifying Existing Screens

Use the Sterling RCP Extensibility Tool to modify or extend existing screens of the Sterling Rich Client application. This tool allows you to modify existing screens by adding or removing text boxes, labels, combo boxes, buttons, table columns, and so forth from the existing forms. You can also add or modify composites and groups. For more information about modifying existing screens, see Section 6.6, "Modifying Existing Screens".

6.5.2.2 Modifying Existing Wizards

You can modify the definition of existing wizards by adding new wizard pages and defining new wizard rules. The wizard rule is used to control the flow of the wizard based on certain criteria. The flow of the wizard depends on the output value of a wizard rule. Use the transition lines to transfer control from one wizard page or rule to another wizard page or rule. The system compares the output of the wizard rule with the transition value. Based on the transition value, the system transfers control to the appropriate wizard page or rule. For more information

about modifying an existing wizard, see Section 6.7, "Modifying Existing Wizards".

6.5.2.3 Creating and Adding New Screens

You can create new Rich Client screens for a Sterling Rich Client application in Eclipse using the Visual Editor (VE). For more information about creating new screens, see Section 6.8, "Creating New Screens".

After creating new screens, you can add these to the Sterling Rich Client application. For more information about adding new screens to a Sterling Rich Client application, see Section 6.9, "Adding New Screens to a Sterling Rich Client Application".

6.6 Modifying Existing Screens

To modify existing screens of a Sterling Rich Client application, perform the following tasks:

- Starting the Sterling RCP Extensibility Tool
- Customizing the User Interface
- Synchronizing Differences
- Building and Deploying Extensions

6.6.1 Starting the Sterling RCP Extensibility Tool

After you set up the development environment, start the Sterling RCP Extensibility Tool. For more information about starting the Sterling RCP Extensibility Tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

6.6.2 Customizing the User Interface

After you start the Sterling RCP Extensibility Tool, you can customize the existing screen by adding or removing text boxes, labels, combo boxes, buttons, table columns, and so forth. You can also add composites and groups to the screen. For more information about customizing screens, see Section 6.20.1, "Customizing Screens".

6.6.3 Synchronizing Differences

Whenever you customize an existing screen, you must synchronize the resource files. For more information about synchronizing differences, see Section 6.20.1.8, "Synchronizing Resource Files".

6.6.4 Building and Deploying Extensions

After you extend the existing screens, make sure that you build and deploy the new extensions. For more information about building and deploying extensions, see Section 6.21, "Building and Deploying RCP Extensions".

6.7 Modifying Existing Wizards

You can modify existing wizards by creating new wizard entities such as wizard page and wizard rule in the new wizard definition. The new wizard definitions are defined in the plug-in project by creating the `<Plug-in_id>_<wizard_name>.ywx` file

Before modifying an existing wizard:

- You must know the form identifier of the wizard you want to extend. Once you have identified the form identifier, define the same form identifier in the extended wizard definition using the `id` attribute of the Wizard element.

- For adding new wizard rules, you must know the namespace for defining new rules and their values. Once you have identified the namespace, define new rules and their values in the `<Plug-in_id>_<wizard_name>.ywx` file.

To get the wizard and namespace information:

1. In the Sterling Rich Client application, navigate to the wizard you want to extend.

2. In the Sterling RCP Extensibility Tool, you can view the screen information. The wizard and the namespace information displays in the screen information window. For more information about viewing screen information, see Section 6.20.1.2, "Viewing Screen Information".

An example of how to extend an existing wizard is shown here:

Let us consider that in the `<Plug-in_id>_<wizard_name>.ycml` file, you have an existing wizard definition defined for the following wizard flow.

WizardRule1 —— Trxn1 : value1 ——▶ WizardPage1 —— Trxn2 ——▶ WizardPage3

You have a wizard rule (WizardRule1) that computes some values and returns a value based on which the control is transferred to a wizard page (WizardPage1). From the WizardPage1 the control is then transferred to another wizard page (WizardPage3).

Now, to extend the above wizard flow, add an inter-mediate wizard page (WizardPage2) between the WizardPage1 and WizardPage3 as shown below:

WizardRule1 —— Trxn1 : value1 ——▶ WizardPage1 —— Trxn2 ——▶ WizardPage2 —— Trxn3 ——▶ WizardPage3
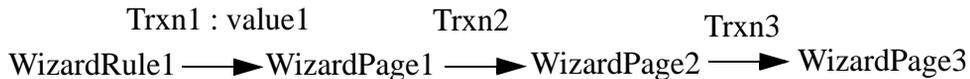
The sample `<Plug-in_id>_<wizard_name>.ycml` XML for the existing wizard definition is as follows:

```
<forms>
<form Id="com.yantra.pca.ycd.rcp.alert.wizard.YCDAlertWizard">
 <namespaces>
  <namespace type="input" name="Rule" templateName="getRule"/>
 <namespaces>
<Wizard>
  <WizardEntities>
    <WizardEntity id="WizRule1">
impl="java:com.yantra.yfc.rcp.wizard.rules.AlertWizRule1"
                type="RULE" xPos="40" yPos="200" start="true">
                <Namespace name="Rule"/>
                <Output value="value1"/>
    </WizardEntity>
    <WizardEntity id="WizPage1">
                    impl="java:com.yantra.yfc.rcp.wizard.pages.AlertWizPage1"
     type="PAGE" xPos="340" yPos="200">
    </WizardEntity>
    <WizardEntity id="WizPage3">
                    impl="java:com.yantra.yfc.rcp.wizard.pages.AlertWizPage3"
     type="PAGE" xPos="140" yPos="200">
    </WizardEntity>
```

```
 </WizardEntities>
 <WizardTransitions>
   <WizardTransition id="WizTrxn1" source="WizRule1">
      <Output target="WizPage1" value="value1"/>
   </WizardTransition>
  <WizardTransition id="WizTrxn2" source="WizPage1"
      target="WizPage3"/>
   <WizardTransitions>
<Wizard>
</form>
</forms>
```

You must create the extended wizard definition in the `<Plug-in_id>_<wizard_name>.ywx` file.

Extending an existing wizard involves:

- Creating the Extended Wizard Definition
- Registering the Wizard Extension File
- Creating the Wizard Entity
- Modifying the Wizard Extension Behavior
- Building and Deploying Extensions

6.7.1 Creating the Extended Wizard Definition

To create the extended wizard definition:

1. Create a new `*.ywx` XML file and save it in the plug-in project that you created while setting up the development environment. For example, `<Plug-in_id><wizard_name>.ywx`.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

2. Start Eclipse SDK.

3. In the navigator view, expand the plug-in project that you created.

4. Right-click on the newly created `*.ywx` file and select Open With > Text Editor from the pop-up menu.

5. Create the Wizards root element.

6. In the `applicationId` attribute, specify the application identifier of the Sterling Rich Client application whose wizard is you want to extend.

For more information about application Id's of a Sterling Rich Client application, see the respective Sterling Rich Client application's documentation.

7. Create the Wizard element under the Wizards root element.

8. In the `id` attribute, specify the form identifier of the wizard you are extending.

9. Create the WizardEntities element under the Wizard element.

10. Create the new wizard entities such as wizard rule and wizard page under the WizardEntities element. For example, create the new wizard page (WizPage2).

11. Create and override the required wizard transitions under the WizardEntities element. For example override the existing transition: WizTrxn2 for transition from WizPage1 to WizPage2 and create a new transition: WizTrxn4 for transition from WizPage2 to WizPage3.

12. Close the Wizard element.

13. Close the Wizards root element.

The sample `<Plug-in_id>_<wizard_name>.ywx` XML file for the extended wizard definition is as follows:

```
<Wizards applicationId="YFSSYS00011">
  <Wizard id="com.yantra.pca.ycd.rcp.alert.wizard.YCDAlertWizard">
   <WizardEntities>
     <WizardEntity id="WizPage2">
                     impl="com.yantra.yfc.rcp.wizard.pages.AlertWizPage2"
               type="PAGE" xPos="340" yPos="200"/>
                </WizardEntities>
     <WizardTransitions>
       <WizardTransition id="WizTrxn2" source="WizPage1"
                             target="WizPage2"/>
       <WizardTransition id="WizTrxn3" source="WizPage2"
                             target="WizPage3"/>
     <WizardTransitions>
  </Wizard>
</Wizards>
```

The `id` attribute of the Wizard entity contains the form identifier of the wizard that you extended. For the new wizard page (WizardPage2), a new WizardEntity element in the WizardEntities element has been created.

In the existing wizard definition, the "WizTrxn2" defines the transition form WizardPage1 to WizardPage3. In the new wizard definition, it is necessary to define transition from WizardPage1 to WizardPage2 and then to WizardPage3. Therefore, you must override the existing "WizTrxn2" transition with the new target, which is WizardPage2. After this, you have to create a new transition "WizTrxn3" to define the transition from WizardPage2 to WizardPage3.

6.7.2 Registering the Wizard Extension File

After creating the extended wizard definition in the newly created `<Plug-in_id>_<wizard_name>.ywx` file, you must register this file with your plug-in. To register your `*.ywx` file, call the registerWizardExtensions() method within the plug-in's constructor. For example,

```
YRCPlatformUI.registerWizardExtensions("<Plug-in_id>_<wizard_name>", ID)
```

where `<Plug-in_id>_<wizard_name>` is the name of your wizard extension file without the ".ywx" extension. ID is a unique identifier of the plug-in that registers this wizard extension file.

> **Note:** Before calling the registerWizardExtensions() method, the plug-in must be registered using the registerPlugin() method of the YRCPlatformUI class. For more information about registering a plug-in, see Appendix B.7, "Registering a Plugin".

6.7.3 Creating the Wizard Entity

You must create the implementation java class for the new wizard entity that you add to the extended wizard definition. It can be a wizard rule or a wizard page. This implementation class is specified in the wizard extension file using the `impl` attribute of the WizardEntity element.

- If you are adding a new wizard page, you must create the implementation java class for the wizard page. For more information about creating new wizard page class, see Section 6.10.2.2.1, "Creating Wizard Page Class".

- If you are adding a new wizard rule, you must create the implementation java class for the wizard rule. For more information about creating new wizard rule, see Section 6.10.2.3, "Creating a Wizard Rule".

6.7.4 Modifying the Wizard Extension Behavior

If you have already created the wizard extension behavior class, and if you are adding a new wizard page, return an instance of the new wizard page in the createPage(String pageIdToBeShown, Composite pnlRoot) method of the wizard extension behavior class. For example:

```
public IYRCComposite createPage(String pageIdToBeShown) {
IYRCComposite page=null;
If(pageIdToBeShown.equalsIgnoreCase(AlertWizPage2.FORM_ID)) { AlertWizPage2 temp
= new AlertWizPage2(new Shell(Display.getDefault(), SWT.NONE);
   page = temp;
}
   return page;
}
```

For more information about creating wizard extension behavior class, see Section 6.20.1.5, "Creating Extension Behavior".

6.7.5 Building and Deploying Extensions

After you extend the existing wizard, make sure that you build and deploy the new extensions. For more information about building and deploying extensions, see Section 6.21, "Building and Deploying RCP Extensions".

6.8 Creating New Screens

After you set up the development environment, start creating the new Rich Client screen. The Sterling RCP provides features that enable you to create Rich Client screens.

To create Rich Client screens, perform the following tasks:

- Creating a Rich Client Platform Composite
- Designing a Rich Client Platform Composite
- Creating Tables
- Binding Classes
- Naming Controls
- Binding Controls
- Localizing Controls
- Theming Controls
- Calling APIs and Services
- Adding New Screens to a Sterling Rich Client application

6.8.1 Creating a Rich Client Platform Composite

The RCP composite consists of:

1. Composite File—The composite java file handles the UI. In the composite java file, write the code for naming, binding, localizing, and theming controls.

2. Behavior File—The behavior java file handles the functionality or behavior of the screen. In the behavior java file, write the code for calling APIs or services and getting or setting the XML model for populating the bound controls.

The Sterling RCP provides following wizards for creating the Rich Client Platform (RCP) composite:

- Sterling RCP Search List Composite Wizard—The Sterling RCP Search List Composite wizard creates a sample search and list screen. Multiple pages of the wizard allows you to name, bind, localize, and theme the controls separately for the search panel and list panel.

- Sterling RCP Composite Wizard—The Sterling RCP Composite wizard creates an empty composite. You need to design the composite by adding appropriate controls as needed. After designing the composite, name, bind, localize, and theme controls that you add to the composite.

6.8.1.1 Creating a Rich Client Platform Composite Using the Sterling RCP Search List Composite Wizard

The Sterling RCP Search List Composite wizard automatically create the composite java file and the behavior java file.

> **Note:** All new commands that you create are stored in the `<Plug-in_id>_commands.ycml` file. All new theme entries that you define are stored in the `<Plug-in_id>_<theme_name>.ythm` file. All new bundle entries that you define are stored in the `<Plug-in_id>_bundle.properties` file.

If you want to call your own API or service instead of Sterling RCP-specific API or service, see Section 6.8.8, "Calling APIs and Services".

To create an Rich Client Platform (RCP) composite using the Sterling RCP Search List Composite wizard:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To store or create the sample search and list screen, right-click on the folder.

4. Select New > Other… from the pop-up menu. The New window displays.

5. From the list of wizards, select Sterling RCP Wizards > Sterling RCP Search List Composite.

6. Click Next. The Search List Screen Preferences window displays.

Table 6–3   Search List Screen Preferences Window

Field	Description
Source Folder	The path of the selected folder automatically displays. You can browse to the folder where you want to store the sample search list screen files.
Package	The name of the package automatically displays.

Table 6–3   Search List Screen Preferences Window

Field	Description
Class Name	By default, the `NewSearchListPanel.java` class file name displays. Enter a new class file name, if necessary.
Resource Bundle	The path of the bundle file that exists in the plug-in project automatically displays, if any. You can browse to the `<Plug-in_id>_bundle.properties` bundle file in which new bundle entries for the search list screen are added.
Commands File	The path of the commands file that exists in the plug-in project automatically displays, if any. You can browse to the `<Plug-in_id>_commands.ycml` commands file in which new commands for the search list screen are added.
Theme File	The path of the theme file that exists in the plug-in project automatically displays, if any. You can browse to the `<Plug-in_id>_<theme_name>.ythm` theme file in which new theme entries for the search list screen are added.
Datatypes Map	Browse to the `yfsdatatypemap.xml` file in which new data types are stored. This file is stored in the location `<YANTRA_ HOME>/Applications/Foundation/template/api/yfsdatatypemap.xml` where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

7. Click Next. The Search Screen Input window displays.

Table 6–4   Search Screen Input Window

Field	Description
Input XML	Select the XML file, which is passed as input to a command or API to populate the search screen.
Search Template	Select the search template file, which is passed as output template to a command or API to populate the list screen.

Table 6–4   Search Screen Input Window

Field	Description
Search Command	Enter the name of the command that is called during the search operation. Depending on the commands in the `<Plug-in_id>_commands.ycml` commands file, you can select the command on which you want the search to be based. You can also create a new command, if needed. If this command is not in the file, entry for the command is created in the command file when the wizard completes.
Group Name	Enter the name of the group that encloses the search screen, if applicable. All controls are positioned in a group box. You can name the group box by entering the name in "Group Name" text box.

8. Click Next. The XML Attribute Selection window displays.
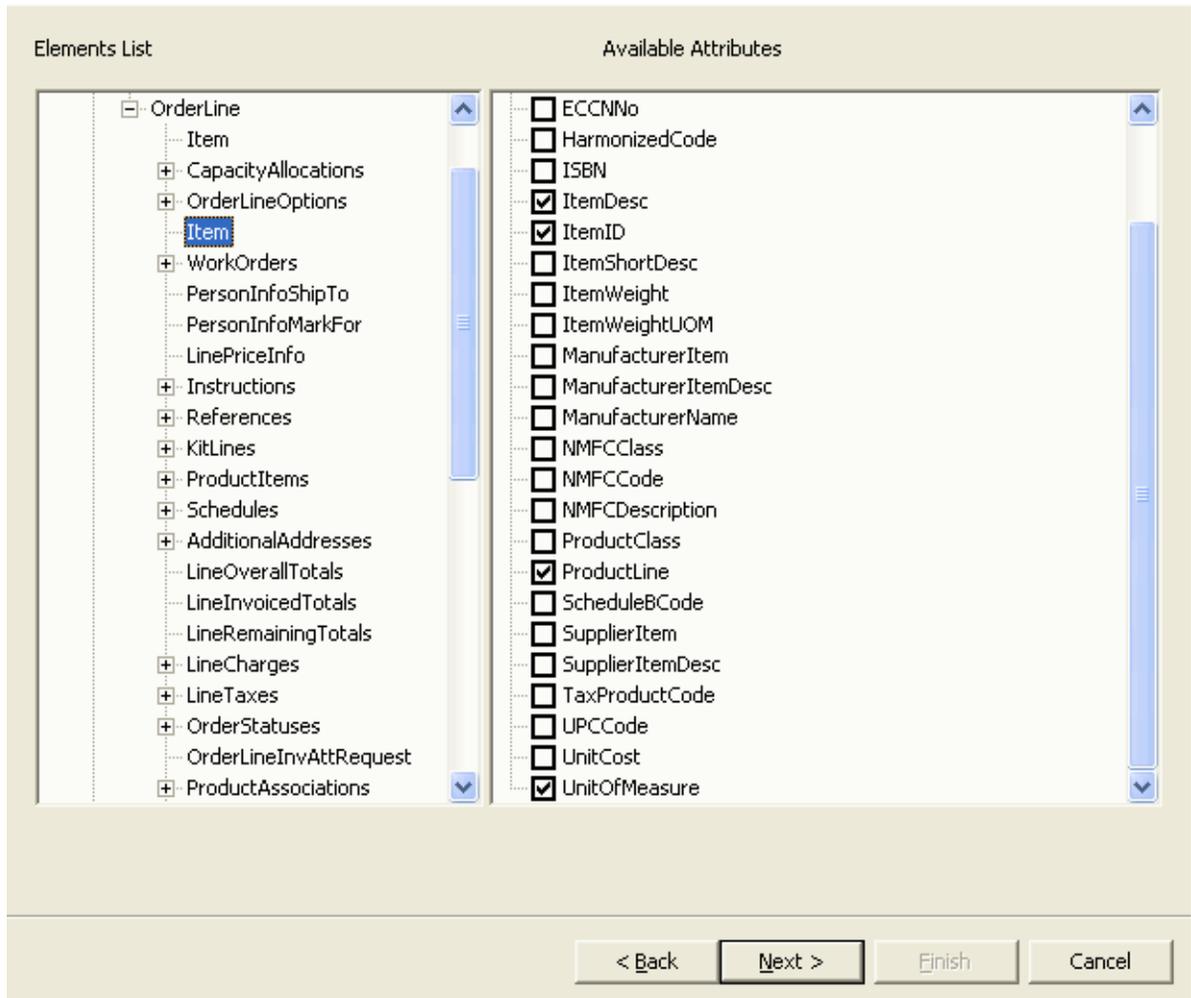
9. Select the XML attributes from the input XML, based on which the search screen is created. The search screen contains fields based on which the search operation is performed. The screen fields are created based on the selected attributes. The Element List panel displays the hierarchy of elements and their attributes for the specified input XML file. The Available Attribute panel displays the list of attributes corresponding to the element selected in the Element List panel.

10. Click Next. The Configure Individual Fields window displays.

!	Name	XML Path	Widget Type	Data Type	Theme	Span	Input Bindi...
	ItemDesc	/Order/OrderLi...	Text	ItemDesc	Text	1	Not Set
	ItemID	/Order/OrderLi...	Text	ItemID	Text	1	Not Set
	ProductLine	/Order/OrderLi...	Text	ProductLine	Text	1	Not Set
	UnitOfMeasure	/Order/OrderLi...	Text	UnitOfMea...	Text	1	Not Set

☑ Set Physical Ordering   Move UP   Move Down   Preview

< Back   Next >   Finish   Cancel

11. Configure the Search screen fields selected in the previous page. Specify the Name, Widget Type, Data Type, Theme, Span, and Input Binding to use on each individual control. You can change the physical ordering of the Search screen controls by clicking the Move Up or Move Down button. This screen also provides visibility to the Search screen fields.

> **Note:** By default, a label is associated with each control unless the control itself is a label.

12. Click Next. The XML Attribute Selection window displays. Figure illustrates XML Attribute Selection window.

13. Select the XML attributes from the input XML. Based on these XML attribute values, the fields in the List screen gets populated. The list screen has a table and all attributes displays in the appropriate columns. The Element List panel displays the hierarchy of elements and their attributes for the specified output template. The Available Attribute panel displays a list of attributes corresponding to the element selected in the Element List panel.

14. Click Next. The List Screen Fields window displays.

15. Configure the List screen fields selected in the previous page. In List Screen Title, enter the title of the List screen. Specify the AttributeBinding, ColumnBinding, ColumnHeader, and width to use on each field. You can change the physical ordering of the List screen controls by clicking the Move Up or Move Down button.

16. Click Next. The Select Template window displays.

17. From the Generator Template combo box, select the template. Based on the template, the Search List screen is created. Each template supports certain configurable attributes. The template is inbuilt with the default layout colspan, width, columnheader, and so forth. To change the configurable attribute, click Configure.

18. Click Finish. The system automatically creates the composite java file such as `NewSearchListPanel.java` and behavior java file such as `NewSearchListPanelBehavior.java` in the specified source folder. These files are stored under the specified package.

6.8.1.2 Creating a Rich Client Platform Composite Using the Sterling RCP Composite Wizard

To create a Rich Client Platform (RCP) composite using the Sterling RCP Composite wizard:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To store the screens, right-click on the folder.

4. Select New > Other… from the pop-up menu. The New window displays.

5. From the list of wizards, select Sterling RCP Wizards category > UI Wizards > Sterling RCP Composite.

6. Click Next. The New Sterling RCP Composite window displays.

Table 6–5   New Sterling RCP Composite Window

Field	Description
Source Folder:	The path of the folder that you selected automatically displays. Click Browse to browse to the source folder where you want to store the composite and behavior java files.

Table 6–5   New Sterling RCP Composite Window

Field	Description
Package:	Enter the name of the package in which you want to store the composite and behavior java files (if necessary). This helps you to easily manage the directory structure of your plug-in project. If not specified, the system automatically creates the composite java file with the default package name.
Composite File Name:	By default, the NewPanel.java composite file name displays. Enter a new composite file name, if necessary.

7. Click Finish. The system creates a composite file and behavior file in the specified source folder. These files are stored in the package that you specified. Figure 6–3 illustrates a typical folder structure that has both Java files stored under the package name.

Figure 6–3   Typical Folder Structure in Eclipse

6.8.2 Designing a Rich Client Platform Composite

You can customize the layout and alignment of your screen as needed. In Visual Editor (Eclipse plug-in), use the Standard Widget Toolkit (SWT) to design UIs.

The Visual Editor enables you to work with these layout managers and easily design your screen. This section explains how to create a simple Search and List screens with an example. Figure 6–4 depicts a sample Search and List screens.

Figure 6–4   Search And List Screen

You can divide the Search and List screen into the following panels:

- Search Criteria panel—This panel contains controls that are used to get input from the user. This may include text boxes, combo boxes, radio buttons, check boxes, and so forth. When you click the search button, the appropriate API is called with contents in the controls as input to the API. The API output is displayed in the Search Results panel.

  For more information about creating the search criteria panel, see Section 6.8.2.1, "Creating the Search Criteria Panel".

- Search Results panel—This panel displays the search results. If the results returns multiple items, you can show the items in a table. Otherwise, you can display data using suitable controls.

  For more information about creating the search results panel, see Section 6.8.2.2, "Creating the Search Results Panel".

6.8.2.1 Creating the Search Criteria Panel

To create the Search Criteria panel for the Search and List screen:

1. Start the Eclipse SDK.

2. In the navigator window, expand the plug-in project that you created.

3. Expand the folder where you have stored the Sterling RCP composite file. For more information about creating the Sterling RCP composite, see Section 6.8.1, "Creating a Rich Client Platform Composite".

4. Right-click on the Sterling RCP composite file and select Open With > Visual Editor from the pop-up menu. The composite file opens in the Visual Editor UI. The Java Beans view automatically opens on the left-hand side of the Eclipse workbench along with other views. Otherwise, manually open the Java Beans View by selecting Window > Show View > Other.… From the list of views under Java, select Java Beans.

5. In the Java Beans view, you can view the hierarchy of SWT containers and controls.

6. In the Properties view, you can view properties and values of the selected containers (composite and group) and controls (labels, text boxes, combo boxes, and so forth).

   You can select containers or controls from the Visual Editor UI or Java Beans View.

7. From the Java Beans view, select "this" composite.

8. From the Properties view, in the `layout` property, select FillLayout from the drop-down list.

9. From the Java Beans view, select "pnlRoot" composite.

10. From the Properties view, select the GridLayout value from the drop-down list for the layout property.

11. From the Palette, click SWT Containers.

12. Select Composite and place it in the pnlRoot composite. The Name pop-up window displays.

13. Enter the name for the Composite. For example, cmpSearchCriteria.

14. From the Properties view, select the GridLayout value from the drop-down list for the layout property.

15. Right-click the cmpSearchCriteria composite, and select the Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays.

Figure 6–5   Customize Layout
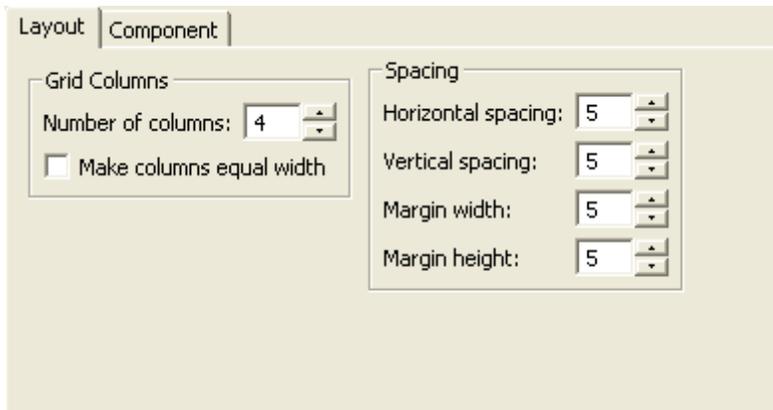
16. In the Grid Columns panel, in Number of columns:, enter 4.

17. Select the Component tab. In the Fill panel, click ⇄ to fill the excess horizontal space.

18. In the Grab Excess panel, click ⇄ to grab the excess horizontal space.

19. Now add various controls to the cmpSearchCriteria composite. For more information about adding controls to the cmpSearchCriteria composite, see Section 6.8.2.1.1, "Adding Controls to the cmpSearchCriteria Composite".

20. Bind the controls to display the required data. For more information about binding controls, see Section 6.8.5, "Binding Controls".

6.8.2.1.1   Adding Controls to the cmpSearchCriteria Composite
To add various controls to the composite follow these steps:

1. From the Palette, click SWT Controls.

2. Select Label and place it in the cmpSearchCriteria composite. The Name pop-up window displays.

3. Enter the name of the Label. For example, lblOrderNo.

4. In the Properties view, enter the text property value as OrderNo.

5. Right-click the lblOrderNo label and select Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

6. Select the Component tab. In the Fill panel, click ⇄ to fill the excess horizontal space. Click ✖.

7. From SWT Controls, select Text and place it after the lblOrderNo label. The Name pop-up window displays.

8. Enter the name for the Text. For example, txtOrderNo.

9. Right-click the txtOrderNo text box and select Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

10. Select the Component tab. In Span panel, in Horizontal, enter 3.

11. In the Fill panel, click ⇄ to fill the excess horizontal space and click ✖.

12. From SWT Controls, select Label and place it after the txtOrderNo text box. The Name pop-up window displays.

13. Enter the name for the Label. For example, lblStatus.

14. In the Properties view, enter the text property value as Status.

15. Right-click the lblStatus label and select Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

16. Select the Component tab. In the Fill panel, click ⇄ to fill the excess horizontal space and click ✖.

17. From SWT Controls, select Combo and place it after the lblStatus label. The Name pop-up window displays.

18. Enter the name for the Combo. For example, cmbStatus.

19. Right-click the cmbStatus composite, and select the Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

20. Select the Component tab. In Grab Excess panel, click ⇄ to grab the excess horizontal space.

21. In the Fill panel, click ⇄ to fill the excess horizontal space. Click ❌.

22. From SWT Controls, select CheckBox and place it after the cmbStatus combo box. The Name pop-up window displays.

23. Enter the name of the CheckBox. For example, chkAcrossEnterprise.

24. In the Properties view, enter the text property value as Across Enterprise.

25. Select the Component tab. In the Fill panel, click ⇄ to fill the excess horizontal space. Click ❌.

26. From SWT Controls, select Button and place it after the chkAcrossEnterprise check box. The Name pop-up window displays.

27. Enter the name of the Button. For example, btnSearch.

28. In the Properties view, enter the text property value as Search.

29. Right-click the btnSearch button and select Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

30. Select the Component tab. In the Alignment panel, click ➡ to right align the btnSearch button.

31. In the Grab Excess panel, click ⇄ to grab the excess horizontal space.

32. From SWT Containers, select Group and place it after the btnSearch command button. The Name pop-up window displays.

33. Enter the name of the Group. For example, grpPaymentType.

34. In the Properties view, select the GridLayout value from the drop-down list for the layout property.

35. Enter the text property value as Payment Type.

36. Right-click the grpPaymentType group and select Customize Layout…
from the pop-up menu. The Customize Layout pop-up window
displays as shown in Figure 6–5.

37. In the Grid Columns panel, in Number of columns:, enter 2.

38. Select the Component tab. In Span panel, in Horizontal, enter 4.

39. From SWT Containers, select RadioButton and place it inside the
grpPaymentType group. The Name pop-up window displays.

40. Enter the name for the RadioButton. For example, rdbtnCheck.

41. In the Properties view, enter the text property value as Check.

42. Add another radio button and enter the name for the RadioButton.
For example, rdbtnCreditCard.

43. In the Properties view, enter the text property value as Credit Card.

44. Click 🖫. The Search Criteria panel gets created as shown:

Figure 6–6   Search Criteria Panel
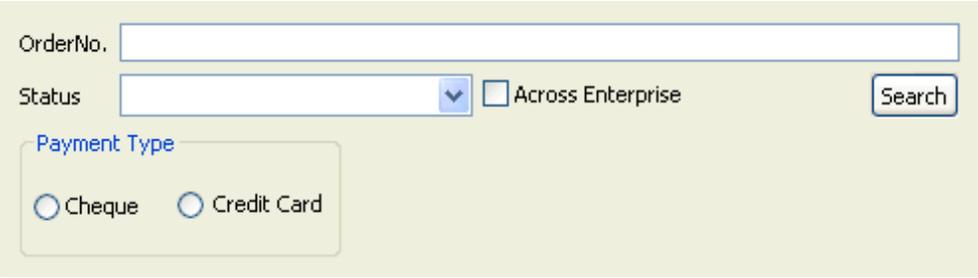
6.8.2.2 Creating the Search Results Panel

To create the Search Results panel for the Search and List screen:

1. From the Palette, click SWT Containers.

2. Select Composite and place it under the pnlRoot composite. The
Name pop-up window displays.

3. Enter the name for the Composite. For example, cmpSearchResult.

4. From the Properties view, select the GridLayout value from the
drop-down list for the cmpSearchResult composite.

5. Right-click the cmpSearchResult composite, and select the Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

6. Select the Layout tab. In Grid Columns panel, in Number of columns, enter 1.

7. Select the Component tab. In Fill panel:

   - Click ⇄ to fill the excess horizontal space.

   - Click ↕ to fill the excess vertical space.

8. In Grab Excess panel:

   - Click ⇄ to grab the excess horizontal space.

   - Click ↕ to grab the excess vertical space.

   - Click ❌.

9. Create a standard table in the cmpSearchResult composite. For more information about creating standard tables, see Section 6.8.3.1, "Creating Standard Tables".

10. Click 💾. The Search Results panel gets created as shown:

Figure 6–7   Search Results Panel

11. Bind the table and table columns with the required data. For more information about binding tables, see Section 6.8.5.3.9, "Binding Tables".

6.8.3 Creating Tables

Sterling RCP supports two types of tables: standard tables and editable tables. As the name suggest, you can modify the data in an editable table, but not in the standard table. This section explains:

- Creating Standard Tables

- Creating Editable Tables

6.8.3.1 Creating Standard Tables

You can create a standard table and add columns to this table.

To create a standard table:

1. From the Palette, click SWT Controls.

2. Select Table and place it in a composite. The Name pop-up window displays.

3. Enter the name for the Table. For example, tblSearchResults.

4. Right-click the tblSearchResults table, and select the Customize Layout… from the pop-up menu. The Customize Layout pop-up window displays as shown in Figure 6–5.

5. Set the layout properties such as Fill, Grab Access, and so forth as needed.

6.8.3.2 Adding Columns to the Standard Table

To add columns to a table:

1. From the Palette, click SWT Controls.

2. Select TableColumn and place it in a table. The Name pop-up window displays. You can add as many columns as you want in a table.

3. Enter the name for each TableColumn that you add to a table. For example, tblcolOrderNo.

4. Bind the table and table columns with the data. For more information about binding a standard table, see Section , "Binding a Standard Table".

6.8.3.3 Creating Editable Tables

To create an editable table:

1. Create a standard table. For more information about creating a standard table, see Section 6.8.3.1, "Creating Standard Tables".

2. To change the standard table to an editable table, associate each table column to a specific cell editor.

> **Note:** You must write the code for creating an editable table in the Sterling RCP composite class.

Create an array of cell editors[] of size that is equal to number of columns. For example:

```
String[] editors = new String[noOfColumns];
```

Sterling RCP supports the following cell editors that are defined in YRCInternalConstants class:

- YRCInternalConstants.YRCComboBoxCellEditor

- YRCInternalConstants.YRCTextCellEditor

- YRCInternalConstants.YRCCheckBoxCellEditor

3. Create a cell editor and associate with a column. This column acts as an editable cell. For example:

```
editors[columnIndex1] = YRCConstants.YRC_COMBO_BOX_CELL_EDITOR;
editors[columnIndex2] = YRCConstants.YRC_TEXT_BOX_CELL_EDITOR;
```

> **Note:** When creating a combo box cell editor you must create a YRCComboBindingData binding object and set the appropriate bindings. For more information about binding a combo box cell editor, see Section , "Binding Combo Box Cell Editors".

4. After creating all cell editors, set the CellTypes for the table with the cell editor array as the input argument. For example:

```
tableBindingData.setCellTypes(editors);
```

5. Bind the table and table columns with the required data. For more information about binding an editable table, see Section , "Binding Editable Tables".

6.8.4 Naming Controls

To name a control, invoke the setName() method on the binding object of that particular control. You must always set a unique name for each control on the screen so that it is easy to refer this control in other files.

To name a control, you must create a binding object.

6.8.4.1 Creating a Binding Object

To create a binding object for naming a control:

Create a new instance of binding class for a specific control. For example, to name a text box, create the following:

```
YRCTextBindingData oData = new YRCTextBindingData();
```

where `YRCTextBindingData` is the class to set bindings for the text box and `oData` is the binding object.

6.8.4.2 **Naming a Control**

Use the binding object that you created to name the control. For more information about creating binding objects, see Section 6.8.4.1, "Creating a Binding Object".

To name a control:

1. Set the name of the control as follows:

```
oData.setName("txtOrderNo");
```

where `txtOrderNo` is the name of the text box.

2. Set the binding data for the control by associating the binding object to the key for that control. For example:

```
txtOrderNo.setData(YRCConstants.YRC_TEXT_BINDING_DEFINATION,oData);
```

where `txtOrderNo` is the reference variable name of the text box, which you specified in the visual editor and `YRCConstants.YRC_TEXT_ BINDING_DEFINATION` is the key used for identifying the text box binding object.

> **Note:** Sterling Commerce recommends that you do not use the same binding object for multiple controls.

If the binding object for a control such as composite or group does not exist, or if you want to name a control without creating the binding object, you can directly set the name for that control using the setData() method. For example,

```
grpSearchCriteria.setData(YRCConstants.YRC_CONTROL_NAME, "grpSearchCriteria");
```

where `grpSearchCriteria` is the reference variable name of the group, which you specified in the visual editor.

6.8.5 **Binding Controls**

Bindings are defined to map an input XML model to the screen and back from the screen to an target XML model.

This section explains the following:

• Binding Classes

- XML Bindings for Different Controls

- Setting Bindings for Different Controls

6.8.5.1 Binding Classes

The Sterling RCP allows you to create binding objects of the following class types for different controls:

- YRCLabelBindingData class for binding labels.

- YRCTextBindingData class for binding text boxes.

- YRCStyledTextBindingData class for binding styledtext components.

- YRCComboBindingData class for binding combo boxes.

- YRCListBindingData class for binding list boxes.

- YRCButtonBindingData class for binding check boxes and radio buttons.

- YRCLinkBindingData class for binding links.

- YRCTableBindingData class for binding tables.

- YRCTblClmBindingData class for binding table columns.

6.8.5.2 XML Bindings for Different Controls

Each control is associated with a set of bindings. The Sterling RCP supports the following types of bindings for the control types:

1. Label—For labels, set the Source Binding.

2. Check Box—For check boxes, set the following bindings:

- Source Binding

- Target Binding

- Checked Binding

- Unchecked Binding

3. Radio Button—For radio buttons, set the following bindings:

- Source Binding

- Target Binding

- Checked Binding

4. Text Box, StyledText component, and Link—For Text Boxes, StyledText components, and links, set the following bindings:

- Source Binding

- Target Binding

5. Combo Box and List Box—For combo boxes and list boxes, set the following bindings:

- Source Binding

- Target Binding

- List Binding

- Code Binding

- Description Binding

6. Table—For Tables, set the following bindings:

- Source Binding

- Target Binding

7. Table Column—For Table columns, set the Attribute Binding.

6.8.5.2.1 Source Binding

Source binding displays XML data in the screen returned by an API by mapping the XML attributes to the screen components. Use the source binding to specify the XML path of an attribute whose value you want to get from an XML model and display in a control. For example, consider the following XML model:

```
<OrderList>
    <Order OrderNo="Y00102495" Status="Accepted">
</OrderList>
```

If you want to get the value of the `OrderNo` attribute from the XML model and display in a text box, set the source binding for the text box as:

```
txtBindingData.setSourceBinding("OrderDetails:OrderList/Order/@OrderNo")
```

where `txtBindingData` is the text box binding object and `OrderDetails` is the namespace of the XML model.

When you set the source binding for a table, specify only the repeating element of the XML model. For example, consider the following XML model:

```
<OrderList>
    <Order OrderNo="Y00102495" Status="Accepted">
    <Order OrderNo="Y00992495" Status="Scheduled">
      <Order OrderNo="Y00990195" Status="Shipped">
</OrderList>
```

Now, set the source binding for the table as:

```
tblBindingData.setSourceBinding("Results:Orderlist/Order")
```

where `tblBindingData` is the table binding object, `Results` is the namespace of the XML model, and `Order` is the repeating element in the XML model.

Multiple Source Bindings

Sterling RCP supports multiple source bindings, which allows you to display values of multiple attributes in the same control. You can separate multiple source bindings by a semicolon. Using the key binding, you can change the format of the multiple source binding values.

Sterling RCP allows you to set multiple source bindings for a control. However, the XML model should have the same namespace for multiple binding attributes.

For example, consider the XML model as specified in Section 6.8.5.2.1, "Source Binding". To display the values of both `OrderNo` and `OrderDate` attributes of the XML model in a text box, do the following:

1. Set the multiple source bindings for the text box as:

```
txtBindingData.setSourceBinding("OrderDetails:OrderList/Order/@OrderNo;Order
Details:OrderList/Order/@Status")
```

where `txtBindingData` is the text box binding object and `OrderDetails` is the namespace of the XML model.

2. Set the key binding for the text box as:

```
txtBindingData.setKey("orderno_and_status_description")
```

where `orderno_and_status_description` is the key.

For more information about the key binding, see Section 6.8.5.2.9, "Key Binding".

3. In the `<Plug-in id>_bundle.properties` file, enter the key value pair (`<key>` = `<value>`) bundle entry for the multiple source binding as:

```
orderno_and_status_description = The Order No. {0} was booked on {1}
```

where `orderno_and_status_description` is the key. `{0}` and `{1}` are the positions of the binding attributes in the XML path.

The value in the text box displays as: The Order No. Y00102495 was booked on 2005-04-07.

6.8.5.2.2   Target Binding

Target binding allows you to create an input XML for an API that contains data entered on the screen. You can use the target binding to specify the XML path of an attribute whose value you want to get from a control and set in an XML model. For example, consider the following XML model:

```
<OrderList>
    <Order OrderNo="Y00102495" Status="Accepted">
</OrderList>
```

If you want to set the value entered in the text box for the `OrderNo` attribute in the XML model, set the target binding for the text box as:

```
txtBindingData.setTargetBinding("OrderDetails:OrderList/Order/@OrderNo")
```

where `txtBindingData` is the text box binding object and `OrderDetails` is the namespace of the XML model.

When you set the target binding for a table, specify only the repeating element of the XML model. For example, consider the following XML model:

```
<OrderList>
    <Order OrderNo="Y00102495" Status="Accepted">
    <Order OrderNo="Y00992495" Status="Scheduled">
      <Order OrderNo="Y00990195" Status="Shipped">
</OrderList>
```

Now, set the target binding for the table as:

```
tblBindingData.setTargetBinding("Results:Orderlist/Order")
```

where `tblBindingData` is the table binding object, `Results` is the namespace of the XML model, and `Order` is the repeating element in the XML model.

Multiple Target Bindings

Sterling RCP supports multiple target bindings allowing you to set the value of the attributes at multiple places in the XML models. This is useful when you want to pass the value of a single control on the screen as an input to multiple APIs. You can also specify multiple target bindings for a control by separating them by a semicolon. For example, consider the following XML models:

OrderListDetails is the namespace of the following model.

```
<OrderList OrderNo="Y00102495" OrderDate="2005-04-07">
    <Order OrderNo="Y00102495" Status="Accepted">
</OrderList>
```

OrderLineDetails is the namespace of the following model.

```
<OrderLine>
    <OrderLineList>
        <Order OrderNo="Y00102495" ItemID="MOUSE"/>
    </OrderLineList>
</OrderLine>
```

If you want to set the value entered in the text box for the `OrderNo` attribute in the XML models, set the target binding for the text box as:

```
txtBindingData.setTargetBinding("OrderListDetails:OrderList/@OrderNo;OrderListDe
tails:OrderList/Order/@OrderNo;OrderLineDetails:OrderLine/OrderLineList/Order/@O
rderNo")
```

where `txtBindingData` is the text box binding object. `OrderListDetails` and `OrderLineDetails` are the namespaces of the XML models.

6.8.5.2.3 Checked Binding

Checked binding is used only for check boxes and radio buttons. Checked binding is used to specify the value based on which radio button gets selected or check box gets checked or unchecked. Use the checked

binding to specify a string to get and set the value of an attribute in an XML model.

When getting the attribute value, the system compares the string value with the attribute value in the XML model. If the value matches, the check box of the corresponding attribute is automatically checked.

When you check a box, the system sets the string value specified in the Checked binding as the attribute value in the XML model.

For example, consider the following XML model:

```
<OrderList>
    <Order OrderNo="Y001" Status="Accepted"
           IsAccrossEnterprise="Y" FromHistory="N"/>
</OrderList>
```

For example, to get and set the value of the IsAccrossEnterprise attribute value as "Y", set the checked binding as follows:

```
btnBindingData.setCheckedBinding("Y")
```

where btnBindingData is the button binding object.

6.8.5.2.4 Unchecked Binding

The unchecked binding is used to specify a string to get and set the value of an attribute in an XML model.

When comparing the value of an attribute, the value of the specified string is compared with the attribute value. If the value matches, the check box of the corresponding attribute gets automatically unchecked.

When setting the value of an attribute, the system sets the string value as the attribute value in the XML model when you uncheck the box.

For example, consider the XML model as specified in Section 6.8.5.2.3, "Checked Binding". If you want to get and set the value of the IsAccrossEnterprise attribute as "N", set the unchecked binding as:

```
btnBindingData.setUnCheckedBinding("N")
```

where btnBindingData is the button binding object.

6.8.5.2.5 List Binding

Use the list binding to specify the XML path of the repeating element to populate the list box or combo box with a list of attribute values.

For example, consider the following XML model:

```
<Order>
    <OrderStatusList>
    <OrderStatus Status="1001" StatusDesc="Created"/>
    <OrderStatus Status="1002" StatusDesc="Packed"/>
    <OrderStatus Status="1003" StatusDesc="Released"/>
      <OrderStatus Status="1004" StatusDesc="Shipped"/>
    </OrderStatusList>
</Order>
```

If you want to populate the list box or combo box with the StatusDesc attribute values, set the list binding as:

```
cmbBindingData.setListBinding("OrderStatusDetails:Order/OrderStatusList/OrderSta
tus")
```

where cmbBindingData is the combo binding object, OrderStatusDetails is the namespace of the XML model, and OrderStatus is the repeating element in the XML model.

> **Note:** You must not specify an XML attribute in the list binding.

6.8.5.2.6 Code Binding

Use the code binding to specify the XML path of an attribute. The value assigned to the code binding attribute is based on the value selected from the list box or combo box.

For example, consider the XML model as specified in Section 6.8.5.2.5, "List Binding". To get the value of the Status attribute based on the value selected for the StatusDesc attribute, set the code binding as:

```
cmbBindingData.setCodeBinding("Status")
```

where Status is the attribute whose value is picked from the XML model based on the value of the StatusDesc attribute, which is specified in the Section 6.8.5.2.7, "Description Binding".

6.8.5.2.7 Description Binding

Use the description binding for displaying the attribute's value on the screen. To display the attribute value, specify the attribute corresponding to the repeating element that you specified in the list binding.

For example, consider the XML model as specified in Section 6.8.5.2.5, "List Binding". If you want to display the value of StatusDesc attribute, then set the Description Binding as:

```
cmbBindingData.setDescriptionBinding("StatusDesc")
```

where cmbBindingData is the combo binding object and StatusDesc is the attribute corresponding to the repeating element specified in the List Binding.

6.8.5.2.8   Attribute Binding

Use the attribute binding to specify the XML path of the attribute whose value you want to display in a table column. For example, consider the following XML model:

```
<OrderList>
    <Order ItemID="MOUSE" ItemDesc="Pointing device"/>
    <Order ItemID="KEYBOARD" ItemDesc="Keyboard Device"/>
    <Order ItemID="PENCIL" ItemDesc="7HB Bold Pencil"/>
      <Order ItemID="PEN" ItemDesc="Super Pen"/>
</OrderList>
```

To display the value of ItemID attribute in the table column:

- Set the source binding for the table as:

  ```
  ItemDetails:OrderList/Order
  ```

  where Order is the repeating element in the XML model.

- Specify the attribute binding as: ItemID.

  where ItemID is the attribute corresponding to the repeating element as specified in source binding.

Multiple Attribute Bindings

Sterling RCP supports multiple attribute bindings, which allows you to display the values of multiple attributes in a table column. You can separate multiple attribute bindings by a semicolon. Using the key binding, you can change the format of the multiple attribute binding values.

Sterling RCP allows you to set multiple attribute bindings for a control. But, the XML model must have the same namespace for multiple binding attributes.

For example, to display the values of `ItemID` and `ItemDesc` attributes in a table column:

- Set the attribute binding for the table column as:

  ```
  ItemID;ItemDesc
  ```

- Set the key binding for the table column as:

  ```
  item_description
  ```

  where `item_description` is the key.

  For more information about key binding, see Section 6.8.5.2.9, "Key Binding".

- In the bundle file, enter the <key> = <value> pair bundle entry for the above specified source binding as:

  ```
  item_description = {0} : {1}
  ```

  where `item_description` is the key. {0} and {1} is the position of the binding attributes in the XML path as specified in the source binding. For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

As a result, the table column displays the value as: MOUSE : Pointing Device.

6.8.5.2.9 Key Binding

Use the key binding to specify a resource bundle key, which you want to use to format and display the XML data within a localizable sentence or combined with another XML data attribute. The key binding is used in conjunction with the source binding or attribute binding as described in the previous sections. For example, consider the following XML model:

```
<OrderList>
     <Order ItemID="MOUSE" ItemDesc="Pointing device"/>
</OrderList>
```

To format the value of the `ItemID` attribute:

- Specify the source binding as:

```
ItemDetails:OrderList/Order/@ItemID
```

- Specify the key binding as: `item_description`

The bundle file contains the following <key>=<value> pair bundle entry
for the above specified key:

```
item_description= The item ordered is : {0}
```

where `item_description` is the key. {0} is the position of the binding
attributes in the XML path.

The value displayed is: The item ordered is MOUSE.

6.8.5.3 Setting Bindings for Different Controls

Consider the following input and target XML models for specifying the
different bindings for controls:

Input XML Model

```
<Order OrderNo="Y001" Status="Included In Shipment"
      IsAccrossEnterprise="Y" FromHistory="N" Link
      Binding="A Link Binding Example : Click Me">
    <OrderLineList>
        <OrderLine ItemID="MOUSE" CodeDescription="First
                   Class" Code="A"/>
        <OrderLine ItemID="PEN" CodeDescription="Second
                   Class" Code="B"/>
        <OrderLine ItemID="PENCIL" CodeDescription="First
                   Class" Code="A"/>
    </OrderLineList>
</Order>
```

Target XML Model

```
<OrderList>
    <Order OrderNo="Y001" Status="Accepted"
          CodeDescription="First Class" Code="A"
          IsAccrossEnterprise="Y" FromHistory="N"/>
    <Order OrderNo="Y002" Status="Released"
          CodeDescription="Second Class" Code="B"
          IsAccrossEnterprise="N" FromHistory="Y"/>
    <Order OrderNo="Y003" Status="Shipped"
          CodeDescription="Third Class" Code="C"
          IsAccrossEnterprise="Y" FromHistory="Y"/>
```

```
</OrderList>
<OrderStatus>
    <Order OrderNo="Y001" Status="Accepted"/>
    <Order OrderNo="Y002" Status="Released"/>
    <Order OrderNo="Y003" Status="Shipped"/>
</OrderStatus>
```

This section explains the following:

- Binding Labels

- Binding Text Boxes

- Binding StyledText Components

- Binding Combo Boxes

- Binding List Boxes

- Binding Check Boxes

- Binding Radio Buttons

- Binding Links

- Binding Tables

6.8.5.3.1   Binding Labels

To set bindings for a label, create a binding object for the label.

Creating a Binding Object

To create a binding object for a label:

Create a new instance of the YRCLabelBindingData binding class. For example:

```
YRCLabelBindingData lblBindingData = new YRCLabelBindingData();
```

where `YRCLabelBindingData` is the class to set bindings for the label and `lblBindingData` is the binding object.

Set Bindings for a Label

1. Set the name of the label using the binding object that you created. For example:

   ```
   lblBindingData.setName("lblOrderNo");
   ```

where `lblOrderNo` is the name of the text box and `lblBindingData` is the binding object.

2. Set the source binding for the label. For example:

```
lblBindingData.setSourceBinding("OrderDetails:Order/@OrderNo");
```

where `OrderDetails` is the namespace of the model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

3. (Optional) Set the multiple source binding for the label. For example:

```
lblBindingData.setSourceBinding("OrderDetails:Order/@OrderNo;Order/@Status")
;
```
where `OrderDetails` is the namespace of the model.

For more information about multiple source binding, see Section , "Multiple Source Bindings".

4. (Optional) Set the key binding for the label. For example:

```
lblBindingData.setKey("order_details");
```

where `order_details` is the key.

> **Note:** If you are specifying multiple source binding for the label, this step is mandatory.

For more information about key binding, see Section 6.8.5.2.9, "Key Binding".

5. (Optional) If you want to display an image for this label, set the server image configuration for the label to display the image from the server. For example:

```
lblBindingData.setServerImageConfiguration(YRCConstants.IMAGE_SMALL);
```

where `IMAGE_SMALL` is the value of the `Name` attribute of the Config element, which is defined in the configuration file. For more information about configuring server images, see the *Sterling Supply Chain Applications Installation Guide*.

6. Set the binding data for the label by associating the binding object to the key. For example:

```
lblOrderNo.setData(YRCConstants.YRC_LABEL_BINDING_DEFINITION,
lblBindingData);
```

where `lblOrderNo` is the reference variable name of the label that you specified in the visual editor and `YRCConstants.YRC_LABEL_BINDING_DEFINATION` is the key used for identifying the label binding object.

6.8.5.3.2 Binding Text Boxes

To set bindings for a text box, you must create a binding object for the text box.

Creating a Binding Object

To create a binding object for a text box:

Create a new instance of the YRCTextBindingData binding class. For example:

```
YRCTextBindingData txtBindingData = new YRCTextBindingData();
```

where `YRCTextBindingData` is the class to set bindings for the text box and `txtBindingData` is the binding object.

Set Bindings for a Text Box

1. Set the name of the text box by using the binding object that you created. For example:

```
txtBindingData.setName("txtOrderNo");
```

where `txtOrderNo` is the name of the text box and `txtBindingData` is the binding object.

2. Set the source binding for the text box. For example:

```
txtBindingData.setSourceBinding("OrderDetails:Order/@OrderNo");
```

where `OrderDetails` is the namespace of the model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

3. (Optional) Set the multiple source binding for the text box. For example:

```
txtBindingData.setSourceBinding("OrderDetails:Order/@OrderNo;Order/@Status")
;
```

where `OrderDetails` is the namespace of the model.

For more information about multiple source binding, see Section , "Multiple Source Bindings".

4. (Optional) Set the key binding for the text box. For example:

```
txtBindingData.setKey("order_details");
```

where `order_details` is the key.

> **Note:** If you are specifying multiple source binding for the text box, this step is mandatory.

For more information about key binding, see Section 6.8.5.2.9, "Key Binding".

5. Set the target binding for the text box. For example:

```
txtBindingData.setTargetBinding("OrderListDetails:OrderList/Order/@OrderNo")
;
```

where `OrderListDetails` is the namespace of the model.

For more information about target binding, see Section 6.8.5.2.2, "Target Binding".

6. (Optional) Set the multiple target binding for the text box. For example:

```
txtBindingData.setTargetBinding("OrderDetails:Order/@OrderNo;OrderStatus/Ord
er/@Status");
```

where `OrderDetails` is the namespace of the model.

For more information about multiple target binding, see Section , "Multiple Target Bindings".

7. Set the binding data for the text box by associating the binding object to the key. For example:

```
txtOrderNo.setData(YRCConstants.YRC_TEXT_BINDING_DEFINATION,txtBindingData);
```

where `txtOrderNo` is the reference variable name of the text box, which you specified in the visual editor and `YRCConstants.YRC_TEXT_BINDING_DEFINATION` is the key used for identifying the text box binding object.

6.8.5.3.3 Binding StyledText Components

To set bindings for a styledtext component, create a binding object for the styledtext component.

Creating a Binding Object

To create a binding object for a styledtext component:

Create a new instance of the YRCStyledTextBindingData binding class. For example:

```
YRCStyledTextBindingData styledTextBindingData = new YRCStyledTextBindingData();
```

where `YRCStyledTextBindingData` is the class to set bindings for the text box and `styledTextBindingData` is the binding object.

Set Bindings for a StyledText Component

1. Set the name of the styledtext component using the binding object that you created. For example:

    ```
    styledTextBindingData.setName("styledTextOrderNo");
    ```

    where `styledTextOrderNo` is the name of the text box and `styledTextBindingData` is the binding object.

2. Set the source binding for the styledtext component. For example:

    ```
    styledTextBindingData.setSourceBinding("OrderDetails:Order/@OrderNo");
    ```

    where `OrderDetails` is the namespace of the model.

    For more information about source binding, see Section Section 6.8.5.2.1, "Source Binding".

3. (Optional) Set the multiple source binding for the styledtext component. For example:

    ```
    styledTextBindingData.setSourceBinding("OrderDetails:Order/@OrderNo;Order/@S
    tatus");
    ```

where `OrderDetails` is the namespace of the model.

For more information about multiple source binding, see Section , "Multiple Source Bindings".

4. (Optional) Set the key binding for the styledtext component. For example:

```
txtBindingData.setKey("order_details");
```

where `order_details` is the key.

> **Note:** If you are specifying multiple source binding for the styledtext component, this step is mandatory.

For more information about key binding, see Section 6.8.5.2.9, "Key Binding".

5. Set the target binding for the styledtext component. For example:

```
styledTextBindingData.setTargetBinding("OrderListDetails:OrderList/Order/@Or
derNo");
```

where `OrderListDetails` is the namespace of the model.

For more information about target binding, see Section 6.8.5.2.2, "Target Binding".

6. (Optional) Set the multiple target binding for the styledtext component. For example:

```
styledTextBindingData.setTargetBinding("OrderDetails:Order/@OrderNo;OrderSta
tus/Order/@Status");
```

where `OrderDetails` is the namespace of the model.

For more information about multiple target binding, see Section , "Multiple Target Bindings".

7. Set the binding data for the styledtext component by associating the binding object to the key. For example:

```
styledTextOrderNo.setData(YRCConstants.YRC_STYLED_TEXT_BINDING_
DEFINATION,styledTextBindingData);
```

where `styledTextOrderNo` is the reference variable name of the styledText component, which you specified in the visual editor and `YRCConstants.YRC_STYLED_TEXT_BINDING_DEFINATION` is the key used for identifying the styledtext component binding object.

6.8.5.3.4 Binding Combo Boxes

To set bindings for a combo box, create a binding object for the combo box.

Creating a Binding Object

To create a binding object for a combo box:

Create a new instance of the YRCComboBindingData binding class. For example:

```
YRCComboBindingData cmbBindingData = new YRCComboBindingData();
```

where `YRCComboBindingData` is the class to set bindings for the combo box and `cmbBindingData` is a binding object.

Set Bindings for a Combo Box

1. Set the name of the combo box using the binding object that you created. For example:

```
cmbBindingData.setName("cmbCode");
```

where `cmbCode` is the name of the combo box and `cmbBindingData` is the binding object.

2. Set the source binding for the combo box. For example:

```
cmbBindingData.setSourceBinding("OrderDetails:Order/OrderLineList/OrderLine/
@Code");
```

where `OrderDetails` is the namespace of the model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

> **Note:** For combo box, source binding is used to specify the default value that should get selected in the combo box. The value of the source binding attribute is compared with the code binding attribute and the corresponding value of the description binding attribute gets selected in the combo box.

3. Set the list binding for the combo box. For example:

```
cmbBindingData.setListBinding("OrderListDetails:OrderList/Order");
```

where `OrderListDetails` is the namespace of the model.

For more information about list binding, see Section 6.8.5.2.5, "List Binding".

4. Set the description binding for the combo box. For example:

```
cmbBindingData.setDescriptionBinding("CodeDescription");
```

For more information about description binding, see Section 6.8.5.2.7, "Description Binding".

5. Set the code binding for the combo box. For example:

```
cmbBindingData.setCodeBinding("Code");
```

For more information about code binding, see Section 6.8.5.2.6, "Code Binding".

6. Set the target binding for the combo box. For example:

```
cmbBindingData.setTargetBinding("OrderListDetails:OrderList/Order/@Code");
```

where `OrderListDetails` is the namespace of the model.

For more information about target binding, see Section 6.8.5.2.2, "Target Binding".

> **Note:** For combo box, target binding is used to specify the attribute whose value is set in the target XML model when user selects a value from the combo box. The value of the code binding attribute is set as the value of the target binding attribute in the target XML model.

7. Set the binding data for the combo box by associating the binding object with the key. For example:

```
cmbCommonCode.setData(YRCConstants.YRC_COMBO_BINDING_
DEFINATION,cmbBindingData);
```

where `cmbCommonCode` is the reference variable name of the combo box, which you specified in the visual editor and `YRCConstants.YRC_COMBO_BINDING_DEFINATION` is the key used for identifying the combo box binding object.

6.8.5.3.5 Binding List Boxes

To set bindings for a list box, create a binding object for the list box.

Creating a Binding Object

To create a binding object for a list box:

Create a new instance of the YRCListBindingData binding class. For example:

```
YRCListBindingData lstBindingData = new YRCListBindingData();
```

where `YRCListBindingData` is the class to set bindings for the list box and `lstBindingData` is a binding object.

Set Bindings for a List Box

1. Set the name of the list box using the binding object that you created. For example:

```
lstBindingData.setName("lstCommonCode");
```

where `lstCommonCode` is the name of the list box and `lstBindingData` is the binding object.

2. Set the source binding for the list box. For example:

```
lstBindingData.setSourceBinding("OrderDetails:Order/OrderLineList/OrderLine/
@Code");
```

where `OrderDetails` is the namespace of the model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

> **Note:** For list box, source binding is used to specify the default value that should get selected in the list box. The value of the source binding attribute is compared with the code binding attribute and the corresponding value of the description binding attribute gets selected in the list box.

3. Set the list binding for the list box. For example:

```
lstBindingData.setListBinding("OrderListDetails:OrderList/Order");
```

where `OrderListDetails` is the namespace of the model.

For more information about list binding, see Section 6.8.5.2.5, "List Binding".

4. Set the description binding for the list box. For example:

```
lstBindingData.setDescriptionBinding("CodeDescription");
```

For more information about description binding, see Section 6.8.5.2.7, "Description Binding".

5. Set the code binding for the list box. For example:

```
lstBindingData.setCodeBinding("Code");
```

For more information about code binding, see Section 6.8.5.2.6, "Code Binding".

6. Set the target binding for the list box. For example:

```
lstBindingData.setTargetBinding("OrderListDetails:OrderList/Order/@Code");
```

where `OrderListDetails` is the namespace of the model.

For more information about target binding, see Section 6.8.5.2.2, "Target Binding".

> **Note:** For list box, target binding is used to specify the attribute whose value is set in the target XML model when user selects a value from the list box. The value of the code binding attribute is set as the value of the target binding attribute in the target XML model.

7. Set the binding data for the list box by associating the binding object with the key. For example:

```
lstCommonCode.setData(YRCConstants.YRC_LIST_BINDING_
DEFINITION,lstBindingData);
```

where `lstCommonCode` is the reference variable name of the list box, which you specified in the visual editor and `YRCConstants.YRC_LIST_BINDING_DEFINITION` is the key used for identifying the combo box binding object.

6.8.5.3.6 Binding Check Boxes

To set bindings for a check box, create a binding object for the check box.

Creating a Binding Object

To create a binding object for a check box:

Create a new instance of the YRCButtonBindingData binding class. For example:

```
YRCButtonBindingData chkBindingData = new     YRCButtonBindingData();
```

where `YRCButtonBindingData` is the class to set bindings for the check box and `chkBindingData` is a binding object.

Set Binding for a Check Box

1. Set the name of the check box using the binding object that you created. For example:

```
chkBindingData.setName("chkAcrossEnterprice");
```

where `chkAccrossEnterprice` is the name of the check box and `chkBindingData` is the binding object.

2. Set the source binding for the check box. For example:

   ```
   chkBindingData.setSourceBinding("OrderDetails:Order/@IsAccrossEnterprice");
   ```

   where `OrderDetails` is the namespace of the model.

   For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

3. Set the target binding for the check box. For example:

   ```
   chkBindingData.setTargetBinding("OrderDetails:Order/@IsAccrossEnterprice");
   ```

   where `OrderDetails` is the namespace of the model.

   For more information about target binding, see Section 6.8.5.2.2, "Target Binding".

4. Set the checked binding for the check box. For example:

   ```
   chkBindingData.setCheckedBinding("Y");
   ```

   When getting the `IsAcrossEnterprise` field value from the input XML model, the string "Y" is compared with the `IsAcrossEnterprise` field value in the input XML model. If the value matches, the check box is automatically checked. When setting the field value in the target XML model, the string "Y" is set as the value for `IsAcrossEnterprise` field when you check the box.

   For more information about checked binding, see Section 6.8.5.2.3, "Checked Binding".

5. Set the unchecked binding for the check box. For example:

   ```
   chkBindingData.setUnCheckedBinding("N");
   ```

   When getting the `IsAcrossEnterprise` field value from the input XML model, the string "N" is compared with the `IsAcrossEnterprise` field value in the input XML model. If the value matches, the check box is automatically unchecked. When setting the field value in the target XML model, the string "N" is set as the value for `IsAcrossEnterprise` field when you uncheck the box.

   For more information about unchecked binding, see Section 6.8.5.2.4, "Unchecked Binding".

6. Set the binding data for the check box by associating the binding object to the key. For example:

```
chkAcrossEnterprice.setData(YRCConstants.YRC_BUTTON_BINDING_
DEFINATION,chkBindingData);
```

where `chkAcrossEnterprice` is the reference variable name of the check box, which you specified in the visual editor and `YRCConstants.YRC_BUTTON_BINDING_DEFINATION` is the key used for identifying the check box binding object.

6.8.5.3.7 Binding Radio Buttons

To set bindings for a radio button, create a binding object for the radio button.

Creating a Binding Object

To create a binding object for a radio button:

Create a new instance of the YRCButtonBindingData binding class. For example:

```
YRCButtonBindingData rdBindingData = new YRCButtonBindingData();
```

where `YRCButtonBindingData` is the class to set bindings for the radio button and `rdBindingData` is a binding object.

Set Bindings for a Radio Button

1. Set the name of the radio button using the binding object that you created. For example:

    ```
    rdBindingData.setName("rdOpen");
    ```

    where `rdOpen` is the name of the radio button and `rdBindingData` is the binding object.

2. Set the source binding for the radio button. For example:

    ```
    rdBindingData.setSourceBinding("OrderDetails:Order/@FromHistory");
    ```

    where `OrderDetails` is the namespace of the model.

    For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

3. Set the target binding for the radio button. For example:

    ```
    rdBindingData.setTargetBinding("OrderDetails:Order/@FromHistory");
    ```

where `OrderDetails` is the namespace of the model.

For more information about target binding, see Section 6.8.5.2.2, "Target Binding".

4. Set the checked binding for the radio button to specify the value used to get the `FromHistory` field value from the input XML model. Set the specified value for the `FromHistory` field in the target XML model. For example:

```
rdBindingData.setCheckedBinding("S001");
```

where `S001` is the value of the `rdOpen` radio button.

For example, if there are three radio buttons, create binding for each of the radio buttons. Set name, source binding and target binding for each radio button. Set the checked binding for each radio button with different values such as S001, S002, and S003. Therefore, when getting the value for a particular field from the input XML model, the value "S001" is compared with the value of that field in the input XML model. If the value matches, then the radio button corresponding to that field is automatically selected. When setting the value in the target XML model, the value "S001" is set as the value for that field in the target XML model when you select the radio button corresponding to that field.

For more information about checked binding, see Section 6.8.5.2.3, "Checked Binding".

5. Set the binding data for the radio button by associating the binding object to the key. For example:

```
rdOpen.setData(YRCConstants.YRC_BUTTON_BINDING_DEFINATION,rdBindingData);
```

where `rdOpen` is the reference variable name of the radio button, which you specified in the visual editor and `YRCConstants.YRC_BUTTON_BINDING_DEFINATION` is the key used for identifying the check box binding object.

6.8.5.3.8 Binding Links

To set bindings for a link, you must create a binding object for the link.

Creating a Binding Object

To create a binding object for a link:

Create a new instance of the YRCLinkBindingData binding class. For example:

```
YRCLinkBindingData linkBindingData = new YRCLinkBindingData();
```

where `YRCLinkBindingData` is the class to set bindings for the link and `linkBindingData` is the binding object.

Set Bindings for a Link

1. Set the name of the link using the binding object that you created. For example:

```
linkBindingData.setName("lnkClickHere");
```

where `lnkClickHere` is the name of the link and `linkBindingData` is the binding object.

2. Set the source binding for the link. For example:

```
linkBindingData.setSourceBinding("OrderDetails:Order/@Binding");
```

where `OrderDetails` is the namespace of the model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

3. Set the binding data for the link by associating the binding object to the key. For example:

```
lnkClickHere.setData(YRCConstants.YRC_LINK_BINDING_
DEFINATION,linkBindingData);
```

where `lnkClickHere` is the reference variable name of the link, which you specified in the visual editor and `YRCConstants.YRC_LINK_ BINDING_DEFINATION` is the key used for identifying the link binding object.

6.8.5.3.9 Binding Tables

Binding a table involves binding each column and setting the column binding object to the table binding object. The Sterling RCP supports two types of tables: standard tables and editable tables. This sections explains the following:

- Binding a Standard Table
- Binding Editable Tables

- Binding Extended Tables

- Binding Extended Editable Tables

Binding a Standard Table

To set bindings for a standard table, you must create a binding object for the standard table. Also, you must create a binding object for a column.

Creating a Binding Object for a Standard Table

To create a binding object for a standard table:

Create a new instance of YRCTableBindingData binding class. For example:

```
YRCTableBindingData tblBindingData = new YRCTableBindingData();
```

where `YRCTableBindingData` is the class to set bindings for the standard table and `tblBindingData` is a binding object.

Creating a Binding Object for a Column

To create a binding object for a column:

Create an array of YRCTblClmBindingData[] with an array size equal to the number of columns in the table. For example:

```
YRCTblClmBindingData clmBindingData[] = new YRCTblClmBindingData[no. of columns
in the table];
```

Set Bindings for a Table and Columns

1. Set the name of the table using the table binding object that you created. For example:

   ```
   tblbBindingData.setName("tblSearchResults");
   ```

   where `tblSearchResults` is the name of the table and `tblbBindingData is the binding object`.

2. Set the name of the table column using the table column binding object that you created. For example:

   ```
   clmBindingData[0].setName("clmItemID");
   ```

   where `tblSearchResults` is the name of the table column and `clmBindingData is the binding object`.

3. Associate the `YRCTblClmBindingData()` attribute to each column. For example:

```
clmBindingData[0] = new YRCTblClmBindingData();
```

4. Set the attribute binding for the column. For example:

```
clmBindingData[0].setAttributeBinding("ItemID");
```

For more information about attribute binding, see Section 6.8.5.2.8, "Attribute Binding".

5. (Optional) Set the multiple attribute binding for the column. For example:

```
clmBindingData[0].setAttributeBinding("ItemID;Code");
```

where `OrderDetails` is the namespace of the model.

For more information about multiple target binding, see Section , "Multiple Attribute Bindings".

6. (Optional) Set the key binding for the column. For example:

```
clmBindingData[0].setKey("item_details");
```

where `item_details` is the key.

> **Note:** If you are specifying multiple attribute binding for the column, this step is mandatory.

For more information about key binding, see Section 6.8.5.2.9, "Key Binding".

7. Set the title of the table column. For example:

```
clmBindingData[0].setColumnBinding("item_id");
```

8. Set the server image configuration for the column to display the image from the server. For example,

```
clmBindingData[0].setServerImageConfiguration(YRCConstants.IMAGE_SMALL);
```

where `IMAGE_SMALL` is the value of the `Name` attribute of the Config element, which is defined in the configuration file. For more

information about configuring server images, see the *Sterling Supply Chain Applications Installation Guide*.

9. To sort a column, set the `SortReqd` attribute value to "true". For example:

```
clmBindingData[0].setSortReqd(true);
```

10. To make a column data localized, set the `DbLocaliseReqd` attribute value to "true". For example:

```
clmBindingData[0].setDbLocaliseReqd(true);
```

For more information localizing the database, see Section 6.2.3.3.1, "Database Localization".

11. Repeat Step 2 to Step 10 to set bindings for all columns in the table.

12. To allow navigation through the keys in a table, set the `KeyNavigationRequired` attribute value to "true". For example:

```
tblBindingData.setKeyNavigationRequired(true);
```

13. To sort a table, set the `SortReqd` attribute value to "true". For example:

```
tblBindingData.setSortRequired(true);
```

14. To filter the table based on some value, set the `FilterReqd` attribute value to "true". For example:

```
tblBindingData.setFilterRequired(true);
```

15. Set the source binding for the column. For example:

```
tblBindingData.setSourceBinding("Results:/OrderLineList/OrderLine");
```

where `Results` is the namespace for this model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

16. Set the column binding data on the table binding data using the setTblClmBindings() method. For example:

```
tblBindingData.setTblClmBindings(clmBindingData);
```

17. (Optional) Use the setLinkProvider() method to create links in the table. The setLinkProvider() method takes the IYRCTableLinkProvider interface as input, which contains two methods getLinkTheme() and linkSelected(). You must implement these methods to create links in the table. The linkSelected() method is called when you select any link in the column. For example:

```
tblBindingData.setLinkProvider(new IYRCTableLinkProvider() {
public String getLinkTheme(Object element, int columnIndex) {
return "TableLink"; }
public void linkSelected(Object element, int columnIndex) {
}});
```

In the getLinkTheme() method, add the logic to set themes for links in a column. This method returns the name of the link theme. If it returns null it is assumed that a link is not required.

In the linkSelected() method, add the logic to perform the required operation, when the link on the table column cell gets selected.

> **Note:** To create links in your table, set the LinkRequired flag of the table column binding object to "true". For example:
>
> ```
> clmBindingData[0].setLinkReqd(true);
> ```
>
> where `clmBindingData` is the object of YRCTblClmBindingData class and `0` is the column index.

18. (Optional) Use the setImageProvider() method to add images for a table column. The setImageProvider() method takes the IYRCTableImageProvider interface as input, which contains getImageThemeForColumn() method. You must implement this method to add images in columns. For example:

```
tblBindingData.setImageProvider(new IYRCTableImageProvider() {
public String getImageThemeForColumn(Object element, int columnIndex) {
return null;
}});
```

In the getImageThemeForColumn() method, add the logic for setting a unique image theme for the table column cell based on some condition. This method returns the unique image theme set. If it returns null, the default image theme is applied.

19. (Optional) Use the setColorProvider() method to set different colors for the table columns. The setColorProvider() method takes the IYRCTableColorProvider interface as input, which contains getColorTheme() method. You must implement this method to provide different colors for the table columns. For example:

```
tblBindingData.setColorProvider(new IYRCTableColorProvider() {
public String getColorTheme(Object element, int columnIndex) {
return null;
}});
```

In the getColorTheme() method, add the logic for setting different colors for the table column cells based on some condition. For example, you may want to set different color for non-editable cells that displays data for the status field, and different color for editable cells that displays data for the amount field. This method returns the name of the color theme. If it returns null, the default color theme is applied.

20. (Optional) Use the setFontProvider() method to set different font types for the table columns. The setFontProvider() method takes the IYRCTableFontProvider interface as input, which contains getFontTheme() method. You must implement this method to provide different colors for the table columns. For example:

```
tblBindingData.setFontProvider(new IYRCTableFontProvider() {
public String getFontTheme(Object element, int columnIndex) {
return null;
}});
```

In the getFontTheme() method, add the logic for setting different font types for the table column cells based on some condition. For example, you may want to set different font type for non-editable cells that displays data for the status field and different font type for editable cells that displays data for the amount field. This method returns the name of the font theme. If it returns null, the default font theme is applied.

21. After setting the binding properties for the YRCTableBindingData object, set the binding data for the table by associating the binding object to the key. For example:

```
tblSearchResult.setData(YRCConstants.YRC_TABLE_BINDING_DEFINATION,
tblBindingData);
```

where `YRCConstants.YRC_BUTTON_BINDING_DEFINATION` is the key used for the table binding object.

Binding Editable Tables

Binding editable tables is same as binding standard tables except that when you bind editable tables, you must handle the editable table columns. Therefore, to bind an editable table, follow the steps as described in Binding a Standard Table.

To handle the editable table columns, use the setCellModifier() method. The setCellModifier() method takes the IYRCCellModifier interface as input, which contains three methods allowModifiedValue(), allowModify() and getModifiedValue(). You must implement these methods to control editable features of different columns in the table. For example:

```
tblBindingData.setCellModifier(new IYRCCellModifier() {
protected boolean allowModify(String property, String value, Element element) {
return true;
}
protected int allowModifiedValue(String property, String value, Element element)
{
return 0;
}
protected String getModifiedValue(String property, String value, Element
element) {
return value;
}});
```

In the allowModify() method, add the logic to check whether you want to allow modifications in an editable cell of a table column. For example, you may want to allow modifications for an editable cell, which displays data for the discount field. This method returns a boolean value, "true" or "false". If the method returns a "false" value, it indicates that modifications are not allowed for that cell.

In the allowModifiedValue() method, add the logic for adding further validation constraints to check whether the new value entered is valid or not. This method returns an integer value. If it returns "0", then the existing value is not replaced with the new value.

In the getModifiedValue() method, add the logic to set the modified value for a cell of a table column that you are currently editing. You can use

this method to update some other property based on the current one or to change the format of the property.

Binding Combo Box Cell Editors

Binding combo box cell editors means binding a combo box inside an editable table. To set bindings for a combo box cell editor, do the following:

1. Create a binding object for the combo box. For more information about creating a binding object, see Creating a Binding Object.

2. Set the list binding, description binding, and code binding for the combo box. For more information about setting these bindings, see Set Bindings for a Combo Box.

> **Note:** Only set the list binding, description binding, and code binding for the combo box.

3. Set the binding data of the table column with the YRCComboBindingData binding object as an argument. For example:

```
clmBindingData[columnIndex].setBindingData(cmbBindingData);
```

where `cmbCommonCode` is the reference variable name of the combo box, which you specified in the visual editor and `YRCConstants.YRC_COMBO_BINDING_DEFINATION` is the key used for identifying the combo box binding object.

Binding Extended Tables

To set bindings for an extended table, you must create a binding object for the extended table.

> **Note:** Make sure that you write the code for binding extended tables in the extension behavior class that you created. In the extension behavior class, override the getExtendedTableBindingData() method. In this method create and return the extended table binding object. For example:
>
> ```
> YRCExtendedTableBindingData extntblBindingData = new
> YRCExtendedTableBindingData("tableSearch");
> ```
>
> ```
> // Create and get the advanced column binding map for the
> extended table.
> ```
>
> ```
> HashMap advclmBindingData = getTableColumnBindingData
> ("tableSearch");
> ```
>
> ```
> extntblBindingData.setTableColumnBindingsMap
> ("advclmBindingData");
> ```
>
> ```
> .
> ```
>
> ```
> .     //Set Bindings for Extended Table and Advanced Columns
> ```
>
> ```
> .
> ```
>
> ```
> return extntblBindingData;
> ```
>
> where tableSearch is the name of the extended table.

Creating a Binding Object for an Extended Table

To create a binding object for an extended table:

Create a new instance of YRCExtendedTableBindingData binding class. For example:

```
YRCExtendedTableBindingData extntblBindingData = new
YRCExtendedTableBindingData();
```

where YRCExtendedTableBindingData is the class to set bindings for the extended table and extntblBindingData is a binding object.

Create a Map of the Advanced Column Binding Data

To create a map of the binding data for the advanced column that you added using the Sterling RCP Extensibility Tool:

Create a new instance of HashMap binding class. For example:

```
HashMap bindingDataMap = new HashMap();
```

where HashMap is the class to create a map of the advanced column binding data and bindingDataMap is the hash map. The HashMap contains the name of the advanced column as the key and the corresponding binding data as the value.

Set Bindings for an Extended Table and Advanced Column

1. Create a binding object for an advanced column by creating a new instance of the YRCTblClmBindingData binding class. For example:

```
YRCTblClmBindingData advclmBindingData = new YRCTblClmBindingData();
```

where YRCTblClmBindingData is the class to set bindings for the advanced column and advclmBindingData is a binding object.

> **Note:** Only if you have added an advanced column through extensibility, you need to create the binding object and set bindings for that advanced column.

2. Set the attribute binding for the advanced column. For example:

```
advclmBindingData.setAttributeBinding("ItemID");
```

For more information about attribute binding, see Section 6.8.5.2.8, "Attribute Binding".

3. (Optional) Set multiple attribute binding for the advanced column. For example:

```
advclmBindingData.setAttributeBinding("ItemID;Code");
```

where OrderDetails is the namespace of the model.

For more information about multiple target binding, see Section , "Multiple Attribute Bindings".

4. (Optional) Set the key binding for the advanced column. For example:

```
advclmBindingData.setKey("item_details");
```

where item_details is the key.

> **Note:** If you specify multiple attribute binding for the column, step is mandatory.

For more information about key binding, see Section 6.8.5.2.9, "Key Binding".

5. Set the server image configuration for the advanced column to display the image from the server. For example,

```
advclmBindingData.setServerImageConfiguration(YRCConstants.IMAGE_SMALL);
```

where IMAGE_SMALL is the value of the Name attribute of the Config element, which is defined in the configuration file. For more information about configuring server images, see the *Sterling Supply Chain Applications Installation Guide*.

6. To sort the advanced column, set the SortReqd attribute value to "true". For example:

```
advclmBindingData.setSortReqd(true);
```

7. To localize the advanced column data, set the DbLocaliseReqd attribute value to "true". For example:

```
advclmBindingData.setDbLocaliseReqd(true);
```

For more information about localizing the database, see Section 6.2.3.3.1, "Database Localization".

8. To filter an advanced column based on some value, set the FilterReqd attribute value to "true". For example:

```
advclmBindingData.setFilterRequired(true);
```

9. Add the advanced column binding data object to the data map object. For example:

```
bindingDataMap.put("extn_AdvClm1", advclmBindingData);
```

where bindingDataMap is the hash map binding object, extn_AdvCml is the name of the advanced column added using the Sterling RCP Extensibility Tool, and advclmBindingData is the advanced column binding object.

10. Repeat Step 1 through Step 9 to set bindings for all advanced columns that you add to the extended table using the Sterling RCP Extensibility Tool.

11. (Optional) To sort an extended table, set the `SortReqd` attribute value to "true". For example:

```
extntblBindingData.setSortRequired(true);
```

12. (Optional) To filter an extended table based on some value, set the `FilterReqd` attribute value to "true". For example:

```
extntblBindingData.setFilterRequired(true);
```

13. Set the source binding for the table. For example:

```
extntblBindingData.setSourceBinding("Results:/OrderLineList/OrderLine");
```

where Results is the namespace for this model.

For more information about source binding, see Section 6.8.5.2.1, "Source Binding".

14. (Optional) Use the setLinkProvider() method to create links in the advanced columns that you added using the Sterling RCP Extensibility Tool. The setLinkProvider() method takes the YRCExtendedTableLinkProvider class as input, which contains two methods getLinkTheme() and linkSelected(). You must implement these methods to create links in the advanced columns of an extended table. The linkSelected() method is called when you select any link in the column. For example:

```
YRCExtendedTableLinkProvider extntblLinkProvider = new
YRCExtendedTableLinkProvider() {
public String getLinkTheme(Object element, String property) {
return "TableLink"; }
public void linkSelected(Object element, String property) {
}};
extntblBindingData.setLinkProvider(extntblLinkProvider);
```

In the getLinkTheme() method, add the logic to set themes for links in a column. This method returns the name of the link theme. If it returns null it is assumed that a link is not required.

In the linkSelected() method, add the logic to perform the required operation, when you click the link in the advanced column cell.

> **Note:** To create links in your extended table, set the
> LinkRequired flag of the advanced column binding object to
> "true". For example:
>
> ```
> advclmBindingData.setLinkReqd(true);
> ```
>
> where advclmBindingData is the advanced column binding
> object.

15. (Optional) Use the setImageProvider() method to add images for an
    advanced column. The setImageProvider() method takes the
    YRCExtendedTableImageProvider class as input, which contains
    getImageThemeForColumn() method. You must implement this
    method to add images in advanced columns. For example:

```
YRCExtendedTableImageProvider extntblImageProvider = new
YRCExtendedTableImageProvider() {
public String getImageThemeForColumn(Object element, String property) {
            if (property.equals("@ItemID")) {
                Element e = (Element)element;
                String strImageTheme = e.getAttribute("TableFilter");
                return strImageTheme;
            } return null;
}};
extntblBindingData.setImageProvider (extntblImageProvider);
```

In the getImageThemeForColumn() method, add the logic for setting
a unique image theme for the advanced column cell based on some
condition. This method returns the unique image theme set. If it
returns null, the default image theme is applied.

16. (Optional) Use the setColorProvider() method to set different colors
    for the advanced columns. The setColorProvider() method takes the
    YRCExtendedTableColorProvider class as input, which contains
    getColorTheme() method. You must implement this method to
    provide different colors for the advanced columns. For example:

```
YRCExtendedTableColorProvider extntblColorProvider = new
YRCExtendedTableColorProvider() {
public String getColorTheme(Object element, String property) {
            if (property.equals("@Price")) {
                Element e = (Element)element;
                int price = YRCXmlUtils.getIntAttribute(e,"Price");
            If (price < 50) {
```

```
                    return "ValidationOK";
              else {
                    return "ValidationERROR";
              }
         } return null;
}};
extntblBindingData.setColorProvider (extntblColorProvider);
```

In the getColorTheme() method, add the logic for setting different colors for the advanced column cells based on some condition. For example, you may want to apply a different color for non-editable cells that displays data for the status field, and a different color for editable cells that displays data for the amount field. This method returns the name of the color theme. If it returns null, the default color theme is applied.

Binding Extended Editable Tables

Binding extended editable tables is same as binding extended tables. The only difference is that when you bind extended editable tables, you must handle the editable advanced columns. Therefore, to bind an extended editable table, follow the steps as described in Binding Extended Tables.

To handle the editable advanced columns, use the setCellModifier() method. The setCellModifier() method takes the YRCExtendedCellModifier class as input, which contains three methods allowModifiedValue(), allowModify(), and getModifiedValue(). You must implement these methods to control editable features of different columns in the table. For example:

```
YRCExtendedCellModifier extntblCellModifier = new YRCExtendedCellModifier() {
protected boolean allowModify(String property, String value, Element element) {
return true;
}
protected int allowModifiedValue(String property, String value, Element element)
{
return 0;
}
protected String getModifiedValue(String property, String value, Element
element) {
return value;
}});
extntblBindingData.setCellModifier (extntblCellModifier);
```

In the allowModify() method, add the logic to check whether you want to allow modifications in an editable cell of an advanced column. For example, you may want to allow modifications for an editable cell, which displays data for the discount field. This method returns a boolean value, "true" or "false". If the method returns a "false" value, it indicates that modifications are not allowed for that cell.

In the allowModifiedValue() method, add the logic for adding further validation constraints to check whether the new value entered is valid or not. This method returns an integer value. If it returns zero (0), the existing value is not replaced with the new value.

In the getModifiedValue() method, add the logic to set the modified value for a cell of an advanced column that you are currently editing. You can use this method to update some other property based on the current one or to change the format of the property.

Binding Combo Box Cell Editors

Binding combo box cell editors indicates binding a combo box inside an editable extended table. To set bindings for a combo box cell editor:

1. Create a binding object for the combo box. For more information about creating a binding object, see Creating a Binding Object.

2. Set the list binding, description binding, and code binding for the combo box. For more information about setting these bindings, see Set Bindings for a Combo Box.

   > **Note:** Only set the list binding, description binding, and code binding for the combo box.

3. Set the binding data of the advanced column with the YRCComboBindingData binding object as an argument. For example:

   ```
   advclmBindingData.setBindingData(cmbBindingData);
   ```

   where cmbBindingData is the combo box binding object and YRCConstants.YRC_COMBO_BINDING_DEFINATION is the key used for identifying the combo box binding object.

4. Add the advanced column binding data object to the advanced column data map object. For example:

   ```
   bindingDataMap.put("extn_AdvClm1", advclmBindingData);
   ```

where bindingDataMap is the hash map binding object, `extn_AdvCml` is the name of the advanced column that you added using the Sterling RCP Extensibility Tool, and advclmBindingData is the advanced column binding object.

6.8.6 Localizing Controls

To localize controls, text, or strings:

- Specify the `<key>=<value>` pair in your `<Plug-in id>_ bundle.properties` file at the plug-in level. Here, key is the resource key and value is the literal displayed for the corresponding locale.

- Replace <value> with the translated value.

For example, to localize a label:

1. Set the key value pair bundle file for the label. For example:

```
Customer_Address=Customer Address
```

where `Customer_Address` is the key and `Customer Address` is the value for the key.

2. Set the text of the label with the key as the input argument. For example:

```
lblCustAdd.setText("Customer_Address");
```

where `lblCustAdd` is the reference variable name of the label, which you specified in the visual editor.

> **Note:** The Sterling RCP automatically localizes labels, buttons, group headers, tab folder items, and table column headers. Therefore, the literals used in the binding object must be resource bundle keys if they need to be translated to different languages.

6.8.7 Theming Controls

For theming controls, define the new theme entries in the `<Plug-in_ id>_<theme_name>.ythm` file. For more information about theming controls, see Section 6.17.1, "Theming Controls".

6.8.8 Calling APIs and Services

To call an API or service:

1. Create a command in the `<Plug-in id>_commands.ycml` file and associate the API or services to be called with the command. Make sure that the code used for calling an API or service is written in the behavior class. For example, to call the getOrderList API, you must create a command with the name as getOrderList and in the APIName attribute enter getOrderList. For more information about creating commands, see Section 6.2.3.14, "Commands".

2. Create a YRCApiContext class object. For example:

   ```
   YRCApiContext context = new YRCApiContext();
   ```

3. Set the command name for the context. For example:

   ```
   context.setApiName("getOrderList");
   ```

   where `getOrderList` is the command name that you created in the `<Plug-in id>_commands.ycml` file.

4. Set the form id for the context. For example,

   ```
   context.setFormId(getFormId());
   ```

5. Set the input XML document that is passed to an API or service. For example:

   ```
   context.setInputXml(getTargetModel("Order").getOwnerDocument());
   ```

   where `Order` is the namespace of the XML model.

6. (Optional) Set the key for the context that you created. For example:

   ```
   context.setUserData("InitialData","1");
   ```

   where `InitialData` is the key and `1` is the value for this key. The value of the key is used to uniquely identify the context. This step is mandatory, if you are calling the same API multiple times. For more information about calling same API multiple times, see Section 6.8.8.1, "Calling the Same API/Service Multiple Times".

7. Call the API or service. For example,

   ```
   callApi(context);
   ```

8. After the API or service call is complete, Sterling RCP calls the handleApiCompletion() method of behavior class to validate the output and process it. Therefore, you can write the API completion logic in this method. For example:

```
public void handleApiCompletion (YRCApiContext context) {
    if(context.getInvokeAPIStatus() < 0) {
        // Add logic for the failure condition
    }
    else {
  if(YRCPlatformUI.equals(context.getApiName(),"getOrderList")) {
setOrderList(context);                                          }
        }
}
```

> **Note:** If an API or service call fails, the Sterling RCP throws an exception.

6.8.8.1 Calling the Same API/Service Multiple Times

The Sterling RCP enables you to call the same API or service multiple tiles. For example, if you want to call the getOrderList API three times with a different input XML model as input to the API:

1. Create three objects of the YRCApiContext class. For example,

```
YRCApiContext context1 = new YRCApiContext();
YRCApiContext context2 = new YRCApiContext();
YRCApiContext context3 = new YRCApiContext();
```

2. Set the same command name for each context. For example,

```
context1.setApiName("getOrderList");
context2.setApiName("getOrderList");
context3.setApiName("getOrderList");
```

3. Set the form id for each context. For example,

```
context1.setFormId(getFormId());
context2.setFormId(getFormId());
context3.setFormId(getFormId());
```

4. Set the different input XML document for each context.

```
context1.setInputXml(getTargetModel("Order").getOwnerDocument());
context2.setInputXml(getTargetModel("OrderDetail").getOwnerDocument());
context3.setInputXml(getTargetModel("OrderList").getOwnerDocument());
```

where `Order`, `OrderDetail`, and `OrderList` are the namespaces of the different XML model.

5. Set the key for each context using the UserData key.

```
context1.setUserData("InitialData","1");
context2.setUserData("InitialData","2");
context3.setUserData("InitialData","3");
```

where `InitialData` is the key and `1`,`2`,and `3` are the values for this key based on the each context. The value of the key is used to uniquely identify each context.

6. Call the API for each context.

```
callApi(context1);
callApi(context2);
callApi(context3);
```

7. In the handleApiCompletion() method, get the context.getUserData() to identify each context. Then, validate and process the output at each API level. For example,

```
public void handleApiCompletion(YRCApiContext context) {
    if(YRCPLatformUI.equals(context.getUserData("InitialData"),"1")) {
//Add your own logic for validating and processing the //output at each API
level.
    }
else if(YRCPLatformUI.equals(context.getUserData("InitialData"),"2")) {
//Add your own logic for validating and processing the //output at each API
level.
    }
else if(YRCPLatformUi.equals(context.getUserData("InitialData"),"3")) {
//Add your own logic for validating and processing the //output at each API
level.
}
```

6.8.8.2 Calling Multiple APIs/Services
The Sterling RCP enables you to call multiple APIs. To call multiple APIs, define multiple commands in the `<Plug-in id>_commands.ycml` file.

> **Note:** Sterling Commerce recommends that you call all
> APIs at the same time to reduce the network traffic.

For example, if there are three combo boxes: cmbStatus, cmbEnterprise,
and cmbCountry, you must call APIs or services to display a list of values
for these combo boxes. For instance, the values displayed for the
cmbStatus combo box depends on the output of the getOrderList API.
The values displayed for the cmbEnterprise combo box depends on the
output of the getShipNodeList API. The values displayed for the
cmbCountry combo box depends on the output of the
getCommonCodeList API output.

Therefore, to call multiple APIs or services:

1. In the `<Plug-in id>_commands.ycml` file, define three commands
   with names as getOrderList, getShipNodeList, and
   getCommonCodeList. Associate an API or service with each of these
   commands using the APIName attribute. Make sure that the code
   used for calling an API or service is written in the behavior class.

   For more information about creating commands, see
   Section 6.2.3.14, "Commands".

2. Create a YRCApiContext class object. For example:

   ```
   YRCApiContext context = new YRCApiContext();
   ```

3. To call multiple APIs, set the command names for the commands that
   you created in the `<Plug-in id>_commands.ycml` file. For example:

   ```
   context.setApiNames(new
   String[]{"getOrderStatusList","getShipNodeList","getCommonCodeList"});
   ```

4. Set the form id for the context. For example,

   ```
   context.setFormId(getFormId());
   ```

5. Set input XMLs for multiple APIs. For example:

   ```
   context.setInputXmls(new
   Document[]{"getOrderStatusList","getShipNodeList","getCommonCodeList"});
   ```

6. (Optional) Set Unique key for multiple commands. For example:

   ```
   context.setUserData("InitialData","1");
   ```

7. Call the API or service. For example:

```
callApi(context);
```

8. Invoke the handleApiCompletion() method to validate and process the output at each API level. You must call this method after executing the callApi() method. For example:

```
public void handleApiCompletion(YRCApiContext context) {

String[] sAPINames = context.getApiNames();
if(YRCPlatform.equals(sAPINames[0],"getOrderStatusList")) {
setOrderList(context);                          }
else if(YRCPlatform.equals(sAPINames[1],"getShipNodeList")) {
setShipNodeList(context); }                                 else
if(YRCPlatform.equals(sAPINames[2],"getCommonCodeList")) {
setCommonCodeList(context); }}
```

6.8.9 Adding New Screens to a Sterling Rich Client application

After creating the new screen, you can add the new screen to the Sterling Rich Client application. For more information about adding new screens to a Sterling Rich Client application, see Section 6.9, "Adding New Screens to a Sterling Rich Client Application".

6.9 Adding New Screens to a Sterling Rich Client Application

You can add the new Rich Client screens to the Sterling Rich Client application:

- Using Pop-up Screens
- Using Menu
- Using Editor

6.9.1 Using Pop-up Screens

You can display the new screen as a pop-up screen, when you click on a button. You need to associate the new screen with the button. To display a new screen as a pop-up screen:

1. Add a new button to an existing screen. For more information about adding new buttons, see Section 6.20.1.4.2, "Adding a Button".

   > **Note:** While adding the new button, make sure that you check the "Validation Required?" box.

2. Synchronize the extension behavior for the screen. For more information about synchronizing extension behavior, see Section 6.20.1.8.4, "Synchronizing Extension Behavior".

3. In the navigator view, expand the plug-in project that you created when setting up the development environment.

4. Expand the package and open the extension behavior class, which you specified in Step 2.

5. In the validateButtonClick() method, add the logic to display the new screen in a pop-up window or dialog window, when you click on the newly added button. For example,

```
ViewOrderDetails screen  = new ViewOrderDetails(new
Shell(Display.getDefault()), SWT.NONE, bindingData, filterObjectList);

YRCDialog oDialog = new YRCDialog(screen,400,400,"OrderDetails",null);
oDialog.open();
```

   where `ViewOrderDetails` is the class name of the screen and `OrderDetails` is the title of the dialog window that displays this screen.

6.9.2 Using Menu

You can display the new screen as a menu item. The menu items are connected to the actions by specifying the action identifier for a specific menu item. Configure the action which gets invoked, when you click on the menu item or a related task. To add new screens to a Sterling Rich Client application menu, define screens in the resources. All resources of Sterling Supply Chain Applications have a set of primary properties that are common to all types of resources. For example, all resources have a Resource ID. These resources are used to define screens. In addition to primary properties, each type of resource has a set of unique properties that is specific to a particular type of resource.

For adding new screens to an application in the resources, define the Resource ID, URL, and Resource Type. The Resource ID is a unique identifier for each resource. The URL contains the RCP ActionId of the class that invokes the screen, which is defined in the `plugin.xml` file.

> **Note:** The action identifiers are not specific to menus. The Related Tasks can also invoke these actions. For more information about RCP actions, see Appendix B.6, "Creating New Actions".

The class that invokes the newly created screen must be created by extending the YRCAction class. In the YRCAction class, the execute() method invokes the action configured by you when you click on a menu item. In the execute() method you can write a code to open the new screen either in a pop-up window or an editor. For more information about opening a screen using a pop-up windows, see Section 6.9.1, "Using Pop-up Screens". For information on how to open a screen in an editor, see Section 6.9.3, "Using Editor".

For more information about defining resources, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

6.9.3 Using Editor

You can display the new screen in an editor when you click on a button or a menu item or a related task. To display a new screen in an editor:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in the Plug-in Manifest Editor, do any of the following:

   • Double-click the `plugin.xml` file.

   • Right-click the `plugin.xml` file and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select
   org.eclipse.ui.editors extension point from the list.

6. Click Finish.

7. Select the org.eclipse.ui.editors extension point. The Extension
   Details panel displays.

8. In the Extension Details panel, enter the properties of
   org.eclipse.ui.editors extension point.

9. Right-click on org.eclipse.ui.editors extension and select New >
   editor. The editor extension element gets created.

10. Select the editor extension element. The Extension Element Details
    panel displays.

11. Enter the properties of the editor extension element.

12. In id*, enter the identifier for the editor.

13. In icon, browse to the path of the icon that you want to associate
    with this editor.

14. In class, to specify the implementation class, do any of the following:

    - Click Browse. The Select Type pop-up window displays. Select the
      class that extends the YRCEditorPart class.

    - Click on the class: hyperlink. The Java Attribute Editor window
      displays.

Table 6–6   Java Attribute Editor Window

Field	Description
Source folder:	The name of the source folder that you selected to store the editor class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the editor class automatically displays. Click Browse to browse to the package where you want to store the editor class.
Name	Enter the name of the editor class.

Table 6–6   Java Attribute Editor Window

Field	Description
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter YRCEditorPart and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the YRCEditorPart superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the YRCEditorPart superclass.
Finish	When you click on this button, the system creates the new editor class in the selected folder or package.

15. Open the newly created editor class in the Java Editor.

16. In the createPartControl() method create and return the instance of the new screen that you created. For example,

```
public void createPartControl(Composite parent) {
ViewOrderDetails screen  = new ViewOrderDetails(parent, SWT.NONE);
return screen;
}
```

17. To open the new screen in the specified editor using the menu item, define a new resource in the resources for the new menu item. For more information about opening new screens using menu, see Section 6.9.2, "Using Menu".

18. In the execute() method of the action set that you associated with the menu item in the previous step do the following:

- create a new input element to pass to the YRCEditorInput object.

- create a new input object to pass to the YRCEditorInput object, if required.

- create a new YRCEditorInput object. Pass the input element and the input object that you created (if required). Also pass the array of strings, which contains the attribute of the input element, and the related task.

- Open the editor that you created for the new screen by passing the Id of the editor to the YRCPlatformUI.openEditor() method.

> **Note:** Make sure that the editor identifier that you pass to the YRCPlatformUI.openEditor() method is same as specified in Step 12.

For example,

```
Element inputElement = YRCXmlUtils.createFromString("<Order
OrderNo=\"YCD001\" />").getDocumentElement();
Object inputObject = new String("");
YRCEditorInput editorInput = new YRCEditorInput(inputElement, inputObject,
new String[]{"OrderNo"}, "YCD_TASK_QUICK_ACCESS");
YRCPlatformUI.openEditor("com.yantra.qa.editors.QAEdito", editorInput);
```

6.10 Creating New Wizards

A wizard is used for any task consisting of many steps, which must be completed in a specific order. A wizard acts as an interface to lead a user through a complex task, using step-by-step pages. It can also be used for the execution of any task involving a sequential series of steps.

Wizard behavior means that each wizard page in a sequence contains a "Next" button, which the user clicks to move to the next wizard page after entering data or configuring information in the current wizard page. If the user decides to go back and change any information entered in a previous wizard page, each wizard page contains a "Previous" button that the user clicks to go back. At the end of the wizard sequence, the user clicks a Finish button to begin the particular process.

To create a wizard do the following:

> **Note:** Before you can start creating wizards, you must set up the development environment. For more information about setting up development environment, see Section 6.4, "Setting Up the Development Environment".

- Creating Wizard Definition
- Creating Wizard Components

6.10.1 Creating Wizard Definition

You can create a new or modify an existing wizard definition by adding wizard entities and wizard transitions. The flow of the wizard depends on the output value of a wizard rule. The wizard definition is created in the `<Plug-in_id>_commands.ycml` file.

> **Note:** You must use a separate `<Plug-in_id>_<wizard_name>.ycml` file for each wizard definition you create.

Creating a new wizard definition involves:

- Opening the Sterling RCP Wizard Editor
- Adding a Wizard Rule
- Adding a Wizard Page
- Adding a Wizard Transition
- Registering Commands File

6.10.1.1 Opening the Sterling RCP Wizard Editor

Sterling RCP Wizard Editor is used for creating or modifying the wizard definition. To open the `<Plug-in_id>_commands.ycml` file in Sterling RCP Wizard Editor:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displayed in the Navigator view.

3. In the navigator window, expand the plug-in project that you created while setting up the development environment. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. Right-click the `<Plug-in_id>_<wizard_name>.ycml` file, select Open With > Sterling RCP Wizard Editor from the pop-up menu.

5. The Sterling RCP Wizard Editor displays. A Palette is available on the right hand side containing the list of tools that can be used to create or modify wizard definition. For example, Marquee, Transition, Rules, and Page.

6.10.1.2 Adding a Wizard Rule

To add a new wizard rule:

1. Open the `<Plug-in_id>_<wizard_name>.ycml` file using the Sterling RCP Wizard Editor. For more information about opening Sterling RCP Wizard Editor, see Section 6.10.1.1, "Opening the Sterling RCP Wizard Editor".

2. From the Palette, click Rules and select Rule.

3. Place the Rule in the Wizard Definition editor where you want to add it.

4. In the properties view, in Id, enter the unique identifier for the wizard rule.

5. In Impl, enter the fully qualified path of the implementation class for this wizard rule. For example,

   ```
   java:com.yantra.pca.ycd.rcp.wizard.rules.NewWizardRule1
   ```

   where `com.yantra.pca.ycd.rcp.wizard.rules` is the package name and `NewWizardRule1` is the wizard rule class name which provides the implementation for this wizard rule.

   In a wizard rule, you can also specify a Greex rule you want to evaluate. To specify the Greex rule: in Impl, enter the relative path of the `*.greex` file. The `*.greex` file contains the Greex rule you want to evaluate. For example,

   ```
   greex:greexRules/test1.greex
   ```

   where `test1.greex` is the name of the Greex file. `greexRules` is the directory in your plug-in project that contains the `*.greex` file.

   > **Note:** You can use only those Greex rules whose return type is either String or Boolean.

   For more information about Greex rule or an advanced XML condition, see the *Sterling Supply Chain Applications Customization Guide*.

6. In Namespaces, enter the namespaces of the XML model based on which the output of a rule is computed. You can enter more than one namespaces for a rule by separating them with a semi-colon. Enter the output values (enter the output for the rule, which is used to

define the flow of the wizard). Based on the values returned by the rule the control is transferred to a wizard entity. You can define more than one output values for a wizard rule by separating them with semi-colon. These namespaces are defined in the `<Plug-in_id>_command.ycml` file. For more information about defining namespaces, see Section 6.13.1, "Defining Namespaces".

7. (Optional) In X and Y, enter the X and Y co-ordinates for this wizard rule. These co-ordinates are relative to the (0,0) co-ordinates of the top left corner.

6.10.1.3 Adding a Wizard Page

To add a new wizard page:

1. Open the `<Plug-in_id>_<wizard_name>.ycml` file using the Sterling RCP Wizard Editor. For more information about opening Sterling RCP Wizard Editor, see Section 6.10.1.1, "Opening the Sterling RCP Wizard Editor".

2. From the Palette, click Page and select Page.

3. Place the Page in the Wizard Definition editor where you want to add it.

4. In the properties view, in Id, enter the unique identifier for the wizard page.

5. In Impl, Enter the fully qualified path of the implementation class for this wizard page that you created. For example,

```
com.yantra.pca.ycd.rcp.wizard.pages.NewWizardPage1
```

where `com.yantra.pca.ycd.rcp.wizard.pages` is the package name and `NewWizardPage1` is the wizard page class name which provides the implementation for this wizard page.

6. (Optional) In X and Y, enter the X and Y co-ordinates for this page. These co-ordinates are relative to the (0,0) co-ordinates of the top left corner.

6.10.1.4 Adding a Wizard Transition

Wizard transition is used to transfer control from one wizard entity to another wizard entity. The wizard transition value is compared with the output of the wizard rule and based on this value the control is

transferred to the next wizard entity. You can define same wizard transition id for multiple wizard transitions. But they must have different values associated with them.

To add a new wizard transition:

1. Open the `<Plug-in_id>_commands.ycml` file using the Sterling RCP Wizard Editor. For more information about opening Sterling RCP Wizard Editor, see Section 6.10.1.1, "Opening the Sterling RCP Wizard Editor".

2. From the Palette, select Transition.

3. Click on the wizard entity from which you want to transfer the control and then click on the wizard entity to which you want to transfer the control.

4. In the properties view, Transition Id, enter the identifier for this wizard transition.

> **Note:** Multiple wizard transitions originating from a wizard rule must have same wizard transition Ids. There can only be at most one transition from the wizard page.

5. In Value, Enter the value for which this transition is to be performed. This value is compared with the output value returned by a wizard rule and depending on this value the control is transferred to the appropriate wizard entity.

> **Note:** Each transition Id should have a unique value associated with it.

6.10.2 Creating Wizard Components

After creating the new wizard definition, you need to create the individual wizard components that provide implementation for the new wizard.

This section explains the following:

- Creating a Wizard
- Creating a Wizard Page

- Creating a Wizard Rule

- Registering Commands File

6.10.2.1 Creating a Wizard

A wizard contains a wizard class and a wizard behavior class.

- Wizard Class—The container class that controls the UI of the wizard.

- Wizard Behavior Class—The container class that controls the behavior of the wizard. Primary function of this class is to display the appropriate wizard pages in a wizard.

This section explains the following:

- Creating Wizard Class

- Creating Wizard Behavior Class

6.10.2.1.1 Creating Wizard Class

To create a wizard class:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displayed in the Navigator view.

3. In the navigator window, expand the plug-in project that you created while setting up the development environment.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. To store the wizard class, right-click on a folder or package and select New > Class from the pop-up menu. The New Java Class window displays.

Figure 6–8   New Java Class window

Table 6–7   New Java Class Window

Field	Description
Source folder:	The name of the source folder that you selected to store the wizard class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the wizard class automatically displays. Click Browse to browse to the package where you want to store the wizard class.
Name	Enter the name of the wizard class.
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter YRCWizard and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the YRCWizard superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the YRCWizard superclass.

5. Click Finish. The system creates the new wizard class in the folder or package selected by you.

6. Open the newly created wizard class in the Java Editor.

7. Right-click in the editor window, select Source > Override/Implement Methods… option from the pop-up menu. The Override/Implement Methods window displays.

8. Select getFormId(), getHelpId(), and createBehavior() methods from the list of methods provided in the YRCWizard class and click OK.

9. Create the field FORM_ID and specify the identifier of the wizard in this field. For example,

```
public static final String FORM_ID =
"com.yantra.pca.ycd.rcp.wizard.NewWizard";
```

Override the getFormId() method and return this form id field.

> **Note:** The identifier specified in the FORM_ID field should be the same form id that you specified in the wizard definition.

10. In the wizard class constructor initialize the wizard by calling the initializeWizard() method. For example,

```
public NewWizard(String wizardId, Composite parent, Object wizardInput, int
style) {
        super(wizardId, parent, wizardInput, style);
        initializeWizard();
    }
```

11. Override the createBehavior() method. Create and return an instance of the wizard behavior class. For example,

```
protected YRCWizardBehavior createBehavior() {
myBehavior = new RCPRIWizardBehavior(this, FORM_ID);
return myBehavior;
}
```

For more information about creating wizard behavior class, see Section 6.10.2.1.2, "Creating Wizard Behavior Class".

6.10.2.1.2 Creating Wizard Behavior Class

To create a wizard behavior class:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displayed in the Navigator view.

3. In the navigator window, expand the plug-in project that you created while setting up the development environment.

For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. To store the wizard behavior class, right-click on a folder or package and select New > Class from the pop-up menu. The New Java Class window displays.

Figure 6–9   New Java Class Window

Table 6–8   New Java Class Window

Field	Description
Source folder:	The name of the source folder that you selected to store the wizard behavior class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the wizard behavior class automatically displays. Click Browse to browse to the package where you want to store the wizard behavior class.
Name	Enter the name of the wizard behavior class.
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter YRCWizardBehavior and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the YRCWizardBehavior superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the YRCWizardBehavior superclass.

5. Click Finish. The system creates the new wizard behavior class in the folder or package selected by you.

6. Open the newly created wizard behavior class in the Java Editor.

7. Right-click in the editor window, select Source > Override/Implement Methods… option from the pop-up menu. The Override/Implement Methods window displays.

8. Select initPage(String) method from the list of methods provided in the YRCWizardBehavior class and click OK.

9. In the initPage(String) method, write the code for performing wizard page specific operations. For example, setting the model, calling API or service, and so forth.

10. In the createPage(String pageIdToBeShown, Composite pnlRoot) method, return an instance of a wizard page corresponding to the pageId. This method is called internally.

```
public IYRCComposite createPage(String pageIdToBeShown, Composite pnlRoot) {
IYRCComposite page=null;
If(pageIdToBeShown.equalsIgnoreCase(NewWizardPage1.FORM_ID)) {
NewWizardPage1 temp = new NewWizardPage1(pnlRoot, SWT.NONE);
   page = temp;
} else if(pageIdToBeShown.equalsIgnoreCase(NewWizardPage2.FORM_ID))
{NewWizardPage2 temp = new NewWizardPage2(pnlRoot,    SWT.NONE);
   page = temp;
}
return page;
}
```

For more information about creating wizard page class, see
Section 6.10.2.2.1, "Creating Wizard Page Class".

6.10.2.2 Creating a Wizard Page

A wizard page contains a wizard page class and a wizard page behavior
class.

- Wizard Page Class—The container class that controls the UI of the
  wizard page to take inputs from the user. In addition, this class takes
  care of binding controls, and so forth.

- Wizard Page Behavior Class—The container class that controls the
  behavior of the wizard page.

This section explains the following:

- Creating Wizard Page Class

- Creating Wizard Page Behavior Class

6.10.2.2.1 Creating Wizard Page Class

To create a wizard page class:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The
   plug-in project is displayed in the Navigator view.

3. In the navigator window, expand the plug-in project that you created
   while setting up the development environment.

   For more information about creating a plug-in project, see
   Section 6.4.3, "Creating a Plug-in Project".

4. To store the wizard page class, right-click on a folder or package and select New > Class from the pop-up menu. The New Java Class window displays.

Source folder:	CustomerOrderManagement	Browse...
Package:	com.yantra.pca.ycd.rcp.wizard	Browse...
☐ Enclosing type:		Browse...

Name: NewWizardPage1

Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static

Superclass: org.eclipse.swt.widgets.Composite Browse...

Interfaces: ① com.yantra.yfc.rcp.IYRCComposite Add...

Remove

Which method stubs would you like to create?
☐ public static void main(String[] args)
☑ Constructors from superclass
☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?
☐ Generate comments

Finish   Cancel

Table 6–9   New Java Class Window

Field	Description
Source folder:	The name of the source folder that you selected to store the wizard page class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the wizard page class automatically displays. Click Browse to browse to the package where you want to store the wizard page class.
Name	Enter the name of the wizard page class.
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter Composite and click OK.
Interfaces:	Click Add, the Implemented Interfaces Selection window displays. In Choose a type, enter IYRCComposite and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the Composite superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the Composite superclass.

5. Click Finish. The system creates the new wizard page class in the folder or package selected by you.

6. Open the wizard page java class in the java editor and design the UI to take the inputs from the user as per the requirements. For more information about designing a RCP composite, see Section 6.8.2, "Designing a Rich Client Platform Composite".

7. In the getFormId() method return the unique FORM_ID of this wizard page.

> **Note:** The string identifier specified in the FORM_ID field should be the same form id that you specified for the wizard page in the wizard definition.

8. In the constructor of the wizard page class, create an instance of wizard page behavior class and store is as a field. For example,

```
public NewWizardPage1(Composite parent, int style) {
super(parent, style);
this.setData("FORMID", FORM_ID);
myBehavior = new NewWizardPage1Behavior(this);
}
```

For more information about creating wizard page behavior class, see Section 6.10.2.2.2, "Creating Wizard Page Behavior Class".

6.10.2.2.2 Creating Wizard Page Behavior Class

To create a wizard page behavior class:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displayed in the Navigator view.

3. In the navigator window, expand the plug-in project that you created. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. To store the wizard page behavior class, right-click on a folder or package and select New > Class from the pop-up menu. The New Java Class window displays.

Source folder:	com.yantra.yfc.rcp.ri	Browse...
Package:	com.yantra.yfc.rcp.ri.wizard	Browse...
☐ Enclosing type:		Browse...

Name: NewWizardPage1Behavior

Modifiers: ⦿ public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static

Superclass: com.yantra.yfc.rcp.YRCWizardPageBehavior   Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?
☐ public static void main(String[] args)
☑ Constructors from superclass
☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?
☐ Generate comments

Finish   Cancel

Table 6–10 New Java Class Window

Field	Description
Source folder:	The name of the source folder that you selected to store the wizard page behavior class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the wizard page behavior class automatically displays. Click Browse to browse to the package where you want to store the wizard page behavior class.
Name	Enter the name of the wizard page behavior class.
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter YRCWizardPageBehavior and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the YRCWizardPageBehavior superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the YRCWizardPageBehavior superclass.

5. Click Finish. The system creates the new wizard page behavior class in the folder or package selected by you.

6. In the initPage(String) method, write the code for performing wizard page specific operations. For example, setting the model, calling API or service, and so forth.

6.10.2.3 Creating a Wizard Rule

A wizard rule contains a wizard rule class. This class performs logical computations to evaluate various output values. Based on these output values flow of the wizard is decided.

To add a new wizard rule:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displayed in the Navigator view.

3. In the navigator window, expand the plug-in project that you created while setting up the development environment. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. To store the wizard rule class, right-click on a folder or package and select New > Class from the pop-up menu. The New Java Class window displays.

Source folder:	CustomerOrderManagement	Browse...
Package:	com.yantra.pca.ycd.rcp.wizard	Browse...
☐ Enclosing type:		Browse...

Name:	NewWizardRule1	
Modifiers:	⦿ public ○ default ○ private ○ protected	
	☐ abstract ☐ final ☐ static	
Superclass:	java.lang.Object	Browse...
Interfaces:	ⓘ com.yantra.yfc.rcp.IYRCWizardRule	Add...
		Remove

Which method stubs would you like to create?

    ☐ public static void main(String[] args)

    ☑ Constructors from superclass

    ☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

    ☐ Generate comments

Finish	Cancel

Table 6–11   New Java Class Window

Field	Description
Source folder:	The name of the source folder that you selected to store the wizard rule class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the wizard rule class automatically displays. Click Browse to browse to the package where you want to store the wizard rule class.
Name	Enter the name of the wizard rule class.
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter Object and click OK.
Interfaces:	Click Add, the Implemented Interfaces Selection window displays. In Choose a type, enter IYRCWizardRule and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the superclass.

5. Click Finish. The system creates the new wizard rule class in the folder or package selected by you.

6. In the execute(HashMap namespaceModelMap) method, write the logic for computing the output value using the model that is passed in the namespaceModelMap parameter and return the output value of the rule. This method is called when the wizard flow needs the output value of this rule. Wizard flow is based on the output of this rule, as defined in the wizard definition. The HashMap contains a list of all namespaces and the corresponding models. These namespaces are defined in the `<Plug-in_Id>_<wizard_name>.ycml` file. For more information about defining namespaces, see Section 6.13.1, "Defining Namespaces".

6.10.2.4 Registering Commands File

After you create the wizard definition in the `<Plug-in_id>_<wizard_ name>.ycml` commands file, you must register this command file with the plug-in project that you created, if required. This is required in order to make your new wizard work according to the flow that you have defined in the commands file. For more information about registering commands file, see Appendix B.8.4, "Registering Commands File".

6.11 Adding New Wizards to a Sterling Rich Client Application

You can add the new wizards to the Sterling Rich Client application in following ways:

- Using Pop-up Screens

- Using Menu

- Using Editor

6.11.1 Using Pop-up Screens

You can display the new wizards as a pop-up screen, when you click on a button. You need to associate the new wizard with the button. To display a new wizard as a pop-up screen:

1. Add a new button to an existing screen. For more information about adding new buttons, see Section 6.20.1.4.2, "Adding a Button".

   > **Note:** While adding the new button, make sure that you check the "Validation Required?" box.

2. Synchronize the extension behavior for the screen. For more information about synchronizing the extension behavior, see Section 6.20.1.8.4, "Synchronizing Extension Behavior".

3. In the navigator view, expand the plug-in project that you created while setting up the development environment.

4. Expand the package and open the extension behavior class, which you specified in Step 2.

5. In the validateButtonClick() method, add the logic to display the new screen in a pop-up window or dialog window, when you click on the newly added button. For example,

```
Object WizardInput = YRCXmlUtils.createfromString("<WizardInput/>");
NewWizard wizard  = new NewWizard(NewWizard.FORM_ID,
Shell(Display.getDefault()), WizardInput, SWT.NONE);
wizard.start();
YRCDialog oDialog = new YRCDialog(wizard,400,400,"AddLine",null);
oDialog.open();
```

6.11.2 Using Menu

You can display the new wizard as a menu item. The menu items are connected to the actions by specifying the action identifier for a specific menu item. Configure the action which gets invoked, when you click on the menu item or a related task. To add new wizards to a Sterling Rich Client application menu, define wizards in the Sterling Supply Chain Applications Resources. All Sterling Supply Chain Applications resources have a set of primary properties that are common to all types of resources. For example, all resources have a Resource ID. These resources are used to define wizards. In addition to primary properties, each type of resource has a set of unique properties that is specific to a particular type of resource.

For adding new wizards to an application in the Sterling Supply Chain Applications Resources, define the Resource ID, URL, and Resource Type. The Resource ID is a unique identifier for each resource. The URL contains the RCP ActionId of the class that invokes the wizard, which is defined in the plugin.xml file.

> **Note:** The action identifiers are not specific to menus. The Related Tasks can also invoke these actions. For more information about RCP actions, see Appendix B.6, "Creating New Actions".

The class that invokes the newly created wizard must be created by extending the YRCAction class. In the YRCAction class, the execute() method invokes the action configured by you when you click on a menu item. In the execute() method you can write a code to open the new wizard either in a pop-up window or an editor. For more information about opening a wizard in a pop-up window, see Section 6.11.1, "Using

Pop-up Screens". For information about opening a wizard in an editor, see Section 6.11.3, "Using Editor".

For more information about defining resources, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

6.11.3 Using Editor

You can display the new wizard in an editor when you click on a button or a menu item or a related task. To display a new wizard in an editor:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in the Plug-in Manifest Editor, do any of the following:

   - Double-click the `plugin.xml` file.

   - Right-click the `plugin.xml` file and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select org.eclipse.ui.editors extension point from the list.

6. Click Finish.

7. Select the org.eclipse.ui.editors extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of org.eclipse.ui.editors extension point.

9. Right-click on org.eclipse.ui.editors extension and select New > editor. The `editor` extension element gets created.

10. Select the `editor` extension element. The Extension Element Details panel displays.

11. Enter the properties of the `editor` extension element.

12. In id*, enter the identifier for the editor.

13. In icon, browse to the path of the icon that you want to associate with this editor.

14. In class, to specify the implementation class, do any of the following:

- Click Browse. The Select Type pop-up window displays. Select the class that extends the YRCEditorPart class.

- Click on the class: hyperlink. The Java Attribute Editor window displays.

Table 6–12   Java Attribute Editor Window

Field	Description
Source folder:	The name of the source folder that you selected to store the editor class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the editor class automatically displays. Click Browse to browse to the package where you want to store the editor class.
Name	Enter the name of the editor class.

Table 6–12   Java Attribute Editor Window

Field	Description
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter YRCEditorPart and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the YRCEditorPart superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the YRCEditorPart superclass.
Finish	When you click on this button, the system creates the new editor class in the selected folder or package.

15. Open the newly created editor class in the Java Editor.

16. In the createPartControl() method create and return the instance of the new wizard that you created. For example,

```
public Composite createPartControl(Composite parent, String task) {
Object WizardInput = YRCXmlUtils.createfromString("<WizardInput/>");
NewWizard wizard  = new NewWizard(NewWizard.FORM_ID, parent, WizardInput,
SWT.NONE);
wizard.start();
return wizard;
}
```

17. To open the new wizard in the specified editor using the menu item, define a new resource in the Sterling Supply Chain Applications resources for the new menu item. For more information about opening new wizards using menu, see Section 6.11.2, "Using Menu".

18. In the execute() method of the action set that you associated with the menu item in the previous step do the following:

- create a new input element to pass to the YRCEditorInput object.

- create a new input object to pass to the YRCEditorInput object, if required.

- create a new YRCEditorInput object. Pass the input element and the input object that you created (if required). Also pass the array

of strings, which contains the attribute of the input element, and the related task.

- Open the editor that you created for the new screen by passing the Id of the editor to the YRCPlatformUI.openEditor() method.

> **Note:** Make sure that the editor identifier that you pass to the YRCPlatformUI.openEditor() method is same as specified in Step 12.

For example,

```
Element inputElement = YRCXmlUtils.createFromString("<Order
OrderNo=\"YCD001\" />").getDocumentElement();
Object inputObject = new String("");
YRCEditorInput editorInput = new YRCEditorInput(inputElement,
inputObject, new String[]{"OrderNo"}, "YCD_TASK_QUICK_ACCESS");
YRCPlatformUI.openEditor("com.yantra.qa.editors.QAEdito", editorInput);
```

6.12 Creating Related Tasks

The Sterling RCP provides the ability to create related tasks by grouping a set of appropriate tasks based on the functionality. You must define the group and category for each related task. All related tasks can belong to multiple categories, but limited to one group. For more information about the YRCRelatedTasks extension point, see Section 6.2.3.5, "What are Related Tasks?".

To create related tasks do the following:

- Extending the YRCRelatedTasks Extension Point
- Extending the YRCRelatedTaskCategories Extension Point
- Extending the YRCRelatedTaskGroups Extension Point
- Extending the YRCRelatedTasksDisplayer Extension Point
- Extending the YRCRelatedTasksExtensionContributor Extension Point

6.12.1 Extending the YRCRelatedTasks Extension Point

The Sterling RCP provides the YRCRelatedTasks extension point for defining related tasks. This extension point needs to be used when you want to display a new Related task on the Related tasks view.

Each related task is associated with a group. You can also define multiple categories for each related task. This extension point is defined in the com.yantra.yfc.rcp plug-in. Any plug-in that is dependent on the com.yantra.yfc.rcp plug-in can extend this extension point to define its own related tasks. The YRCRelatedTasks extension has an extension element called tasks. The tasks extension element also has an extension element called task.

Prior to implementing this extension point the following information is required:

- Category Information—When a new related task is being associated to the active task running in the current editor, you need to know the categories which the active task is interested in, so that the new related task can be shown on the Related tasks view. Once you have identified the category id to which the new related task should belong to, you must define the same category definition using the YRCRelatedTaskCategories extension point. Also, make sure that the new related task is defined in the above category.

  > **Note:** You can only add new related task to an existing category.
  >
  > You must define this category in your `plugin.xml` file, and make sure that the new related task is defined under this category.

- Group Information—To display a new related task, you can either use an existing group or create a new group.

  To get the category and the group information for adding the new related task to a existing task:

1. Navigate to the task you want to extend, in the Sterling Rich Client application.

2. Through the Sterling RCP Extensibility Tool view the screen information. The category and group information is displayed in the

screen information window. For more information about viewing screen information, see Section 6.20.1.2, "Viewing Screen Information".

To extend the YRCRelatedTasks extension point:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in the Plug-in Manifest Editor, do any of the following:

   - Double-click the `plugin.xml` file.

   - Right-click the `plugin.xml` file and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select com.yantra.yfc.rcp.YRCRelatedTasks extension point from the list.

6. Click Finish.

7. Select the com.yantra.yfc.rcp.YRCRelatedTasks extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of YRCRelatedTasks extension point.

9. Right-click on com.yantra.yfc.rcp.YRCRelatedTasks extension and select New > tasks. The `tasks` extension element gets created.

10. Select the tasks extension element. The Extension Element Details panel displays.

11. In the Extension Element Details panel, enter the properties of the tasks extension element.

12. To create a new task extension element, right-click on tasks extension you created and select New > task. The task extension element gets created. You can create multiple `task` elements under the tasks extension element.

13. In the Extension Details panel, enter the properties of the task extension element.

14. In permissionId, enter the resource identifier from the Sterling Supply Chain Applications resources that provide implementation for checking permissions to perform the related task. For example, a customer support representative may not have permissions to change the price of an item.

15. In actionId, enter the identifier of the action that gets invoked when you click on the related task. The action class of the actionId that you specified should extend the YRCRelatedTaskAction class.

16. In groupId, enter the identifier of the group to which the related task belongs to. The groups are defined by extending the YRCRelatedTaskGroups extension point. For more information about extending the YRCRelatedTaskGroups extension point, see Section 6.12.3, "Extending the YRCRelatedTaskGroups Extension Point".

17. Set the `isExtension` property to "true" if you want to mark the related task as a extended related task.

> **Note:** Whenever you set the value of the `isExtension` property to "true" for a related task, it indicates that you want to open the extended related task in an existing editor. Therefore, you must define an extension contributor to open the extended related task in an existing editor. For more information about extending the YRCRelatedTaskExtensionContributor extension point, see Section 6.12.5, "Extending the YRCRelatedTasksExtensionContributor Extension Point".

18. Set the `filterRequired` property to "true", if you want to filter related tasks based on custom criteria. For example, the Cancel Order related task should not be displayed after you ship an order.

19. To create a new categories extension element, right-click on the related task for which you want to define the category and select New > categories. The categories extension element gets created.

20. Select the categories extension element. The Extension Element Details panel displays.

21. In the Extension Element Details panel, enter the properties of the categories extension element.

22. To create a new category extension element, right-click on categories extension element that you created and select New > category. The category extension element gets created. Just as Sterling RCP supports defining multiple categories for each related task, you can define multiple category extension elements under the categories extension element.

23. In the Extension Details panel, enter the properties of the category extension element.

24. In id, enter the identifier of the category to which the related task belongs to. These categories are defined by extending the YRCRelatedTaskCategories extension point. For more information about extending the YRCRelatedTaskCategories extension point, see Section 6.12.2, "Extending the YRCRelatedTaskCategories Extension Point".

6.12.2 Extending the YRCRelatedTaskCategories Extension Point

The Sterling RCP provides the YRCRelatedTaskCategories extension point for defining categories, which can contain multiple related tasks from multiple groups. This extension point is defined in the com.yantra.yfc.rcp plug-in. Any plug-in that is dependent on the com.yantra.yfc.rcp plug-in can extend this extension point to define its own categories for the related tasks. The YRCRelatedTaskCategories extension has an extension element called categories. The categories extension element also has an extension element called category.

To extend the YRCRelatedTaskCategories extension point:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment.

For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the plugin.xml file in Plug-in Manifest Editor, do any of the following:

- Double-click the `plugin.xml` file.

- Right-click the `plugin.xml` file, and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select com.yantra.yfc.rcp.YRCRelatedTaskCategories extension point from the list.

6. Click Finish.

7. Select the com.yantra.yfc.rcp.YRCRelatedTaskCategories extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of the YRCRelatedTaskCategories extension point.

9. To create a new categories extension element, right-click on com.yantra.yfc.rcp.YRCRelatedTaskCategories extension and select New > categories. The categories extension element gets created.

10. Select the categories extension element. The Extension Element Details panel displays.

11. In the Extension Element Details panel, enter the properties of the categories extension element.

12. To create a new category extension element, right-click on categories extension element that you created in the previous step and select New > category. The category extension element gets created. You can create multiple category elements under the categories extension element.

13. In the Extension Details panel, enter the properties of the category extension element.

14. To create a new tasks extension element, right-click on `category` extension element that you created and select New > tasks. The tasks extension element gets created.

15. Select the tasks extension element. The Extension Element Details panel displays.

16. In the Extension Element Details panel, enter the properties of the tasks extension element.

17. To create a new task extension element, right-click on tasks extension that you created in the previous step and select New > task. The task extension element gets created. You can create multiple task elements under the tasks extension element. You can associate multiple related tasks to the same category.

18. In the Extension Details panel, enter the properties of the task extension element.

19. In id, enter the id of the related task that you want to have in this particular category. These categories are defined by extending the YRCRelatedTaskCategories extension point. For more information about extending the YRCRelatedTaskCategories extension point, see Section 6.12.2, "Extending the YRCRelatedTaskCategories Extension Point".

6.12.3 Extending the YRCRelatedTaskGroups Extension Point

The Sterling RCP provides the YRCRelatedTaskGroups extension point for defining group for a set of related tasks. This extension point is defined in the com.yantra.yfc.rcp plug-in. Any plug-in that is dependent on the com.yantra.yfc.rcp plug-in can extend this extension point to define its own groups for the related tasks. The YRCRelatedTaskGroups extension has an extension element called groups. The groups extension element also has a extension element called group.

To extend the YRCRelatedTaskGroups extension point:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment.

For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the plugin.xml file in Plug-in Manifest Editor, do any of the following:

- Double-click the plugin.xml file.

- Right-click the plugin.xml file, and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select com.yantra.yfc.rcp.YRCRelatedTaskGroups extension point from the list.

6. Click Finish.

7. Select the com.yantra.yfc.rcp.YRCRelatedTaskGroups extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of the YRCRelatedTaskGroups extension point.

9. To create a new `group` extension element, right-click on com.yantra.yfc.rcp.YRCRelatedTaskGroups extension and select New > groups. The groups extension element gets created.

10. Select the groups extension element. The Extension Element Details panel displays.

11. In the Extension Element Details panel, enter the properties of the groups extension element.

12. To create a new group extension element, right-click on groups extension element that you created and select New > group. The group extension element gets created. You can create multiple group elements under the groups extension element.

13. In the Extension Details panel, enter the properties of the group extension element.

14. In sequence, enter the number to indicate that the groups should display in the ascending order of the sequence number in the Related Tasks view. The groups are displayed in the ascending order of their sequence number.

6.12.4 Extending the YRCRelatedTasksDisplayer Extension Point

The Sterling RCP provides the YRCRelatedTasksDisplayer extension point for specifying the class that implements the com.yantra.yfc.rcp.IYRCRelatedTasksDisplayer interface.

This extension point is used to display the Related tasks view, implement this extension point only if you need to override the way the current view is displayed.

This extension point is defined in the com.yantra.yfc.rcp plug-in. Any plug-in that is dependent on the com.yantra.yfc.rcp plug-in can extend this extension point to provide its own implementation. The YRCRelatedTasks element has an extension element called relatedTasksDisplayer.

To extend the YRCRelatedTasksDisplayer extension point:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in Plug-in Manifest Editor, do any of the following:

   - Double-click the `plugin.xml` file.

   - Right-click the `plugin.xml` file, and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select com.yantra.yfc.rcp.YRCRelatedTasksDisplayer extension point from the list.

6. Click Finish.

7. Select the com.yantra.yfc.rcp.YRCRelatedTasksDisplayer extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of the YRCRelatedTasksDisplayer extension point.

9. To create a new relatedTasksDisplayer extension element, right-click on com.yantra.yfc.rcp.YRCRelatedTasksDisplayer extension and select New > relatedTasksDisplayer. The relatedTasksDisplayer extension element gets created.

10. Select the relatedTasksDisplayer extension element. The Extension Element Details panel displays.

11. To specify the implementation class, do any of the following

- Click Browse. The Select Type pop-up window displays. Select the class that implements the com.yantra.yfc.rcp.IYRCRelatedTasksDisplayer interface. The specified class must return the list of all the related tasks that you want to display in a panel as ArrayList.

- Click on the class* hyperlink. The Java Attribute Editor window displays.

  - Enter the name of the class that implements the com.yantra.yfc.rcp.IYRCRelatedTasksDisplayer interface.

  - Click Finish. The new class gets automatically created.

6.12.5 Extending the YRCRelatedTasksExtensionContributor Extension Point

The Sterling RCP provides the YRCRelatedTasksExtensionContributor extension point for specifying the class that implements the com.yantra.yfc.rcp.IYRCRelatedTasksExtensionContributor interface. Use this extension point only when you want to open a newly created related task within an application shipped editor. This extension contributor is called when a extended related task needs to be executed in an application shipped editor.

Prior to implementing this extension point the following information is required:

- Editor Information—To open the newly created related task within an application shipped editor, you need to know the identifier of that particular editor.

  To get the editor information for opening the new related task in the application shipped editor:

1. Navigate to the task you want to extend, in the Sterling Rich Client application.

2. Through the Sterling RCP Extensibility Tool view the screen information. The editor information is displayed in the screen information window. For more information about viewing screen information, see Section 6.20.1.2, "Viewing Screen Information".

This extension point is defined in the com.yantra.yfc.rcp plug-in. Any plug-in that is dependent on the com.yantra.yfc.rcp plug-in can extend

this extension point to provide its own implementation for extended related tasks. The YRCRelatedTasksExtensionContributor element has an extension element called relatedTasksExtensionContributor.

To extend the YRCRelatedTasksExtensionContributor extension point:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created.

    For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in Plug-in Manifest Editor, do any of the following:

    - Double-click the `plugin.xml` file.

    - Right-click the `plugin.xml` file, and select Open With > Plug-in Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select com.yantra.yfc.rcp.YRCRelatedTasksExtensionContributor extension point from the list.

6. Click Finish.

7. Select the com.yantra.yfc.rcp.YRCRelatedTasksExtensionContributor extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of the YRCRelatedTasksExtensionContributor extension point.

9. To create a new relatedTasksExtensionContributor extension element, right-click on com.yantra.yfc.rcp.YRCRelatedTasksExtensionContributor extension and select New > relatedTasksExtensionContributor. The relatedTasksExtensionContributor extension element gets created. You can create multiple relatedTasksExtensionContributor elements but make sure that for one relatedTasksExtensionContributor extension element you specify a unique editor. Otherwise, the system randomly selects an extension contributor from the specified extension contributors.

10. Select the relatedTasksExtensionContributor extension element. The Extension Element Details panel displays.

11. In editorId, specify the Id of the editor in which the extensible related tasks need to be opened. You can use the Sterling RCP-provided editor or your own custom editor to open the related task. For one relatedTasksExtensionContributor elements, you can specify only one editor Id.

12. To specify the implementation class, do any of the following

   - Click Browse. The Select Type pop-up window displays. Select the class that implements the com.yantra.yfc.rcp.IYRCRelatedTasksExtensionContributor interface.

   - Click on the class* hyperlink. The Java Attribute Editor window displays.

     – Enter the name of the class, that implements the com.yantra.yfc.rcp.IYRCRelatedTasksExtensionContributor interface.

     – Click Finish. The new class gets automatically created.

13. Override the createPartControl(Composite parent, YRCEditorInput editorInput, YRCRelatedTask currentTask) and return the panel to open the current editor. For example,

```
public Composite createPartControl(Composite parent, YRCEditorInput
editorInput, YRCRelatedTask currentTask) {
YCDAlertScreen NewScreen  = new YCDAlertScreen(parent, SWT.NONE);
       return NewScreen;
}
```

6.13 Creating Commands

You can create commands to call APIs or services to retrieve data in the required format. To create commands at the form level, use the `<Plug-in id>_commands.ycml` command file. Each form is a self-contained panel in itself. A self-contained panel has its own behavior class that extends the YRCBehavior class. Therefore, you must specify the identifier of the form in the `Id` attribute of the `form` element.

> **Note:** In case of wizards, although wizard page has its own behavior, it is not a self-contained panel. This is because the wizard page behavior is internally dependent on the wizard behavior. Therefore, to create commands for a wizard page, you must create commands at the wizard level. You must specify the identifier of the wizard in the `Id` attribute of the `form` element.

The various attributes of command element are as following:

Table 6—13   Command Element's Attribute List

Field	Description
Name	Specify a unique name for the command. Command names are unique across the system and redefining a command with the same name overrides an existing definition. For more information about overriding commands, see Section 6.13.2, "Overriding Commands".
APIName	Specify the name of an API or service. Associate each command you create with an API or service name.
APIType	Specify the API type. The valid values are "API" and "SERVICE".
outputNamespace	Specify the namespace of the output template. This namespace is defined in the namespaces element. For more information about defining namespaces, see Section 6.13.1, "Defining Namespaces".
inputNamespace	Specify the namespace of the input XML model. This namespace is used to get the target XML model. For more information about getting the target XML model, see Section 6.8.8, "Calling APIs and Services".
	NOTE: This target XML model is passed as the input XML model to the command only when the input XML model for a command is not set.

Table 6–13   Command Element's Attribute List

Field	Description
URL	If you want to invoke your own API instead of APIs or services of Sterling Supply Chain Applications, specify the URL path of the server in the URL attribute. This URL must contain the value of the Name attribute of the Config element from the *.ycfg file. The complete path of the URL is defined in *.ycfg file. For more information about defining server URL in the *.ycfg file, see the *Sterling Supply Chain Applications Installation Guide*.
prototype	Set the value of prototype attribute equal to "true" to run the commands in prototype mode. In prototype mode, the application uses XMLs stored in the prototype folder on the client machine as an output of an API. For more information about prototype mode, see Section 6.2.3.9, "What is Prototype Mode?".
version	The Sterling RCP also supports versioning of APIs to ensure backward compatibility. Specify the version of the API that you want to call in the version attribute.

The following code is from a typical *.ycml file that is used to create commands:

```
<forms>
    <form Id = "com.yantra.order.capture.ui.screens.OrderSearchandList">
        <commands>
            <command Name="getOrderDetails"
                     APIName="getOrderDetails"
                     APIType="API"
                     outputNamespace="OrderDetails"
                     inputNamespace=""
                     URL="LOCAL"
                     prototype="true"
                     version=""/>
                <command Name = "getOrderList"
                    APIName = "getOrderList"
                    APIType="API"
                    outputNamespace="OrderList"
```

```
                        inputNamespace=""
                        URL=""
                        prototype=""
                        version=""/>
          </commands>
     </form>
</forms>
```

> **Note:** Sterling Commerce recommends that you do not
> make changes to the configuration file shipped with
> Sterling Supply Chain Applications. Always create your own
> configuration file or use the default configuration file that
> gets created whenever you create a new Sterling RCP
> plug-in.

Every plug-in must invoke the command files during plug-in initialization
to register its own set of commands. For more information about
registering a commands file, see Appendix B.8.4, "Registering
Commands File".

6.13.1 Defining Namespaces

Namespaces are defined to uniquely identify an XML model. Use the
`<Plug-in id>_commands.ycml` file to define namespaces. You can define
namespaces at the form level. Specify the unique identifier of the form in
the Id attribute of the form element. The various attributes of namespace
element are as following:

Table 6–14   Namespace Element's Attribute List

Field	Description
name	Specify a unique name for the namespace.
type	Specify the type of namespace, depending on whether the template is to be used as input or output. For example, "input" or "output". NOTE: If you are creating the namespaces for the wizard rules: • Specify type attribute as "input" if you want to take inputs from the user as the target model of the screen. • Specify type attribute as "output" if you want to populate the controls with the values from an existing model.
templateName	Specify the name of the XML file that is to be picked from the server. For example, getOrderDetails. The system searches for this file on the server in the template/<Plug-in_id>/<form_id>/namespaces directory

The following code is from a typical *.ycml file that is used to define namespaces:

```
<forms>
    <form Id = "com.yantra.order.capture.ui.screens.OrderSearchandList">
        <commands>
            <command Name="getOrderDetails"
                    APIName="getOrderDetails"
                    APIType="API"
                    Namespace="OrderDetails"
                    URL=""
                    prototype=""
                    version=""/>
        </commands>
        <namespaces>
            <namespace name="OrderDetails"
                    type="output"
                    templateName="getOrderDetails"/>
            <namespace name="OrderList"
                    type="output"
```

```
                                         templateName="getOrderList"/>
                    </namespaces>
              </form>
       </forms>
```

6.13.2 Overriding Commands

The Overriding Commands feature enables a user to call its custom API instead of APIs provided by Sterling Supply Chain Applications for a particular form. To override a command, you must enter your own custom API name and use the same form Id and command name.

6.14 Template Merging

The Sterling RCP allows you to merge the input and output templates as per your needs. Template merging can be used to get some additional data from an API or service. For example, you may want to get the values of additional attributes from an API or service. All the templates that are shipped with Sterling Supply Chain Applications are stored on the server in the namespaces folder of the Sterling RCP plug-in and PCA plug-in directories.

PCA templates are located at:

```
<YANTRA_HOME>/Applications/<PCA_Application_
Name>/template/<PCA_plug-in_id>/<form_id>/namespaces
```

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

To extend the PCA templates, put the extended templates for the PCA in:

```
<YANTRA_HOME>/Applications/<PCA_Application_
Name>/template/<PCA_plug-in_id>/<form_id>/namespaces/extn
```

Sterling RCP templates are located at:

```
<YANTRA_
HOME>/Applications/Foundation/template/com.yantra.yfc.rcp/name
spaces
```

> **Note:** You cannot extend the Sterling RCP templates.

For APIs or services for which no form id is specified, the output templates are stored in the following folder of the Sterling RCP plug-in:

```
<YANTRA_HOME>/Applications/Foundation/template/<Plug-in_
id>/namespaces
```

You can create new output templates and store them in the following folder of the Sterling RCP plug-in:

```
<YANTRA_HOME>/Applications/Foundation/template/<Plug-in_
id>/<form_id>/namespaces/extn
```

> **Note:** During the creation of the Runtime, merging of the application happens. As a result, all template directories mentioned above are copied to the `<YANTRA_ HOME>/Runtime/template` folder. For more information about creating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

For example, let us consider the following `getOrderLineDetails` output template:

```
<OrderLine>
    <OrderLineList>
        <Order OrderNo="Y00102495" ItemID="MOUSE"/>
    </OrderLineList>
</OrderLine>
```

To add a new attribute called Status to the OrderNo element in this output template:

- Create the XML file with the same name as the existing output template and store it in the `template/<Plug-in_id>/<form_ id>/namespaces/extn` folder of the Sterling Supply Chain Applications PCA plug-in. For example, `getOrderLineDetails.xml`.

- Add the new attribute called Status in the Order element. Only add the additional attributes that you require. The new output template looks like:

```
<OrderLine>
    <OrderLineList>
        <Order Status=" "/>
    </OrderLineList>
</OrderLine>
```

As a result, the new getOrderLineDetails output template looks like:

```
<OrderLine>
  <OrderLineList>
    <Order OrderNo="Y00102495" ItemID="MOUSE" Status=" "/>
  </OrderLineList>
</OrderLine>
```

6.15 Adding New Related Tasks and Hiding Existing Related Tasks

Adding New Related Tasks
You can add new related tasks to the Rich Client application by extending the following extension points provided by the Sterling RCP.

- YRCRelatedTasks
- YRCRelatedTaskCategories
- YRCRelatedTaskGroups
- YRCRelatedTasksDisplayer
- YRCRelatedTasksExtensionContributor

For more information about creating related tasks, see Section 6.12, "Creating Related Tasks".

Hiding Existing Related Tasks
You can also hide the related tasks from the screen by removing the related tasks from the list of related tasks that is specified in the YRCRelatedTasksDisplayer extension point. For more information about

YRCRelatedTasksDisplayer extension point, see Section 6.12.4, "Extending the YRCRelatedTasksDisplayer Extension Point".

6.16 Registering and Using the Shared Tasks

This section explains how to register and use the shared tasks in your application.

6.16.1 Registering Shared Tasks

You can register the new shared tasks with Sterling RCP plug-in using the YRCSharedTasks extension point.

To register the new shared tasks:

1. Start Eclipse SDK.

2. In the navigator view, expand the plug-in project that you created while setting up the development environment.

   For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in the Plug-in Manifest Editor, do any of the following:

   - Double-click the `plugin.xml` file.

   - Right-click the `plugin.xml` file, and select Open With > Plug-in Manifest Editor.

4. Click the Extensions tab.

5. Click Add. From the New Extension window, select com.yantra.yfc.rcp.YRCSharedTasks extension point from the list.

6. Click Finish.

7. Select the com.yantra.yfc.rcp.YRCSharedTasks extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of the YRCSharedTasks extension point.

9. In id*, enter the unique identifier for the shared task. This shared task identifier should be unique across all applications or plug-ins.

10. In name*, enter the name for the shared task.

11. In description*, enter the description of the shared task.

12. In class*, specify the implementation class for the shared task.

To specify the implementation class, do any of the following:

- Click Browse. The Select Type pop-up window displays. Select the class to use to extend the YRCSharedTask class.

- Click the class* hyperlink. The Java Attribute Editor window displays.

Table 6–15   Java Attribute Editor Window

Field	Description
Source folder:	The name of the source folder that you selected to store the shared task class automatically displays. Click Browse to browse to the folder that you want to specify as the source folder.
Package:	The name of the package that you selected to store the shared task class automatically displays. Click Browse to browse to the package where you want to store the shared task class.
Name	Enter the name of the shared task class.

Table 6–15   Java Attribute Editor Window

Field	Description
Superclass:	Click Browse, the Superclass Selection window displays. In Choose a type, enter YRCSharedTask and click OK.
Constructors from superclass	Check this box. The system automatically creates the constructor for the YRCSharedTask superclass.
Inherited abstract methods	Check this box. The system automatically adds the abstract methods inherited by the YRCSharedTask superclass.
Finish	When you click on this button, the system creates the new shared task class in the selected folder or package.

13. Open the newly created editor class in the Java Editor and implement the abstract methods of the YRCSharedTask class.

6.16.2 Using Shared Tasks

You can invoke a shared task by clicking a button, menu item, or so forth. You can invoke a shared task by calling the launchSharedTask(String taskId, Element input) method provided by the YRCPlatformUI utility class of the Sterling RCP.

To invoke a shared task within an application, you must know the complete details of the shared task such the identifier of the shared task, structure of the input XML template, and structure of the output XML template.

To view the shared task information:

1. Navigate to the Sterling Rich Client application.

2. In the Sterling RCP Extensibility Tool, you can view the shared task information.

For more information about viewing the shared task information, see Section 6.20.1.11, "Viewing Shared Tasks".

After getting the required information invoke the shared task by calling the launchSharedTask(String taskId, Element input) method. For example:

```
YRCPlatformUI.launchSharedTask("com.yantra.rcp.SharedTask1",input);
```

where `com.yantra.rcp.SharedTask1` is the identifier of the shared task that you want to invoke and `input` is an input XML element that exist in the input XML to the shared task.

The YRCPlatformUI class provides more methods, which you can call to invoke a particular shared task. For example, launchSharedTask(String taskId), launchSharedTask(Composite parent, String taskId), and so forth.

6.17 Theming the Sterling Rich Client Application

For theming the Sterling Rich Client application, define the new theme entries in the `<Plug-in_id>_<theme_name>.ythm` theme file. After you register the theme file, it is loaded using the user-defined locale. For more information about registering the theme file, see Appendix B.8.2, "Registering Theme File". The system loads all theme entries into a common repository and automatically applies them to the controls on the UI. The last theme definition that is loaded overrides the previous theme definitions.

To define the new theme entries for theming the Sterling Rich Client application:

1. Before you can start theming your Sterling Rich Client application, you must set up the development environment. For more information about setting up the development environment, see Section 6.4, "Setting Up the Development Environment".

2. In the navigator window, expand the plug-in project that you created.

3. Open the `*.ythm` file in the text editor.

4. Create the root element Theme.

5. In the `id` attribute, specify the unique identifier for the theme.

6. Create ThemeEntry element under the Theme element.

7. In the `Name` attribute specify the unique name for this theme entry.

8. Create the Font element under ThemeEntry and set the its attributes. For Font element attribute list, see Table 6–16.

Table 6–16   Font Element Attribute List

Attribute	Description
Name	Specify the name of the font you want to use. For example, Tahoma, Courier, Arial, and so forth.
Height	Specify the height of the font.
Style	Specify the font style that you want to use. For example, NORMAL, BOLD, ITALIC, and so forth.

9. Create the BackgroundColor element under ThemeEntry and set the its attributes. For BackgroundColor element attribute list, see Table 6–17.

Table 6–17   BackgroundColor Element Attribute List

Attribute	Description
Red	Specify the decimal color code for the red color. Valid values range from 0 to 255.
Green	Specify the decimal color code for the green color. Valid values range from 0 to 255.
Blue	Specify the decimal color code for the blue color. Valid values range from 0 to 255.

10. Create the ForegroundColor element under ThemeEntry and set the its attributes. For ForegroundColor element attribute list, see Table 6–17.

11. Create the Image element under the ThemeEntry element, if applicable.

12. In the `Path` attribute, specify the path of the image you want to display.

> **Note:** You can create multiple ThemeEntry elements to define themes for various resources such as control text, user info, error text, error icons, logos, and so forth.

13. Rename the *.ythm file to: <file_name>_<theme_name>.ythm. For example, comapp_jade.ythm.

where comapp is the <file_name> and jade is the <theme_name>.

14. Register the theme file in the plugin java file of the plug-in project using the registerTheme() method. For example,

```
YRCPlatformUI.registerTheme("<file_name>_<themename>", ID);
```

The sample theme entries from the *.ythm file is given below:

```
<Theme id="jade">
    <ThemeEntry Name="Label">
       <Font Name="Tahoma" Height="9" Style="NORMAL"/>
          <ForegroundColor Red="0" Green="0" Blue="0"/>
          <BackgroundColor Red="245" Green="245" Blue="245"/>
    </ThemeEntry>
    <ThemeEntry Name="Text">
     <Font Name="Tahoma" Height="8" Style="NORMAL"/>
     <ForegroundColor Red="0" Green="0" Blue="0"/>
     <BackgroundColor Red="255" Green="255" Blue="255"/>
  </ThemeEntry>
   <ThemeEntry Name="Table">
      <Font Name="Tahoma" Height="8" Style="NORMAL"/>
      <BackgroundColor Red="245" Green="245" Blue="245"/>
      <ForegroundColor Red="0" Green="0" Blue="0"/>
  </ThemeEntry>
   <ThemeEntry Name="ErrorColor">
      <Font Name="Tahoma" Height="10" Style="BOLD"/>
      <ForegroundColor Red="255" Green="0" Blue="0"/>
      <BackgroundColor Red="245" Green="245" Blue="245"/>
  </ThemeEntry>
   <ThemeEntry Name="ErrorIcon">
      <Image Path="/icons/error.gif"/>
  </ThemeEntry>
   <ThemeEntry Name="HeaderLogo">
      <Image Path="/icons/yantra_header.jpg"/>
  </ThemeEntry>
</Theme>
```

6.17.1 Theming Controls

For theming controls, define the theme entries in the `<Plug-in_id>_`
`<theme_name>.ythm` file at the plug-in level. For example, let us consider
that you have created a new label and you want to have a specific font
and color for that label. To set a theme for the label:

1. Define entries in the theme file for the label. For example:

```
<Theme id="sapphire">
    <ThemeEntry Name="MyLabel">
        <Font Height="8" Name="Tahoma" Style="NORMAL"/>
        <ForegroundColor Blue="0" Green="0" Red="0"/>
        <BackgroundColor Blue="245" Green="245" Red="245"/>
    </ThemeEntry>
</Theme>
```

where `id` attribute is the unique identifier for the `<Plug-in_id>_`
`<theme_name>.ythm` file. The `Name` attribute indicates the name of the
theme entry, which is used for theming controls.

> **Note:** The theme file corresponding to the theme
> specified within the user configuration is loaded. For
> example, if you log on to the Sterling Rich Client
> application as user that is configured to use the theme
> with id as "sapphire", then the theme file with id
> "sapphire" gets loaded.
>
> Therefore, if you are creating new screens and adding new
> entries for the "sapphire" theme, the `Id` attribute of this
> extension theme file should be "sapphire".

2. Set the binding data for the control by associating the binding object
    with the key. For example,

    ```
    lblDate.setData(YRCConstants.YRC_CONTROL_CUSTOMTYPE,"MyLabel");
    ```

    where `lblDate` is the reference variable name of the label, which you
    specified in the visual editor, `YRCConstants.YRC_CONTROL_`
    `CUSTOMTYPE` is the key used for identifying the custom theme entry,
    and `MyLabel` is the name of the ThemeEntry element in the theme
    file.

6.18 Adding or Removing Menus

Menu configuration contains the standard Sterling Supply Chain Applications resources and also extended resources that you define when configuring resources.

All menus are grouped into a menu group. The default menu group contains the standard menu configuration of the Sterling Supply Chain Application Consoles, which is linked to the default Administrator user. When creating your own users, you can reuse this menu group or create a new menu group. The custom menus may contain different menu items.

For more information about adding or removing menus from the screen, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

6.19 Setting the Extension Model

Extension model is set to populate the newly added fields on the form with the required data. Extension model must be used in case you are not getting the required data from the existing model or existing APIs or services called on the screen.

Before you set the extension model, do the following:

- Creating Commands—In the `<Plug-in_id>_commands.ycml` file, create new commands for calling an API or service for a screen. For more information, about creating commands, see Section 6.13, "Creating Commands".

- Defining Namespaces—In the `<Plug-in_id>_commands.ycml` file, define the new namespaces for a screen. For more information about defining namespaces, see Section 6.13.1, "Defining Namespaces".

> **Note:** All the new namespaces that you define must start with "Extn_".

After creating new commands and namespaces for a screen, call an API or service. After API or service call completes, call the setExtensionModel() method to populate the newly added fields on the screen. You must pass the namespace of the model and the target element as arguments to the setExtensionModel() method.

> **Note:** Use the setExtensionModel() method only if the a specific API is not returning the required data for the newly added field and hence you want to call your own API.

6.20 Using Sterling RCP Extensibility Tool

Sterling RCP Extensibility Tool allows you to extend the Sterling Rich Client UI by adding new controls, modifying existing controls, and so forth. The Sterling RCP Extensibility tool allows you to add UI controls such as labels, text boxes, combo boxes, list boxes, and so forth. The Sterling RCP Extensibility Tool provides the ability to change properties of the existing Sterling RCP-provided fields. You can also add new fields and specify layout, bindings, and theme properties for these fields. The Sterling RCP Extensibility Tool also allows you to synchronize the differences in theme entries, bundle entries, and templates.

> **Note:** There are some screens in a Sterling Rich Client application that cannot be extended using the Sterling RCP Extensibility Tool. When you perform any operation on such screens using the Sterling RCP Extensibility Tool, the message "Cannot extend an Inextensible Screen" displays.

6.20.1 Customizing Screens

Before customizing any screen, you must load the extension file. For more information about loading extension file, see Section 6.20.1.1.1, "Loading Extension File".

Using the RCP Extensibility Tool, you can customize screens by:

- Starting the Rich Client Platform Extensibility Tool
- Viewing Screen Information
- Viewing Control Information
- Adding New Fields
- Creating Extension Behavior
- Modifying Existing Fields
- Modifying or Deleting the Newly Added Fields

- Synchronizing Resource Files

- Showing Hidden and Disabled Fields

- Adding Secure APIs

- Viewing Shared Tasks

6.20.1.1 Starting the Rich Client Platform Extensibility Tool

Before you start extending the screens, you need to start the RCP Extensibility Tool.

To extend a screen:

1. From the Sterling Rich Client application's menu bar, select File > Extend. The RCP Extensibility Tool opens.

2. Click 🖳 to load the extension file, if applicable.

   For more information about loading extension file, see Section 6.20.1.1.1, "Loading Extension File".

3. Click 🟢 to start extending the screen.

6.20.1.1.1 Loading Extension File

Before you extend or modify the existing forms using the RCP Extensibility Tool, you must load `<Plug-in id>_extn.yuix` extension file in the RCP Extensibility Tool. To open the RCP Extensibility Tool, from the menu bar, select File > Extend. All registered extension files automatically gets loaded in the tool, and displays in the bottom panel. If the extension files are unregistered, you must register them and then load the files prior to making changes to the form. For more information about registering extension files, see Appendix B.8.5, "Registering Extension File".

To load the extension files:

1. From the menu bar, select File > Extend. The RCP Extensibility Tool opens.

2. Click 🖳. The Open dialog displays.

3. Select the `<Plug-in id>_extn.yuix` extension file.

If you do not load `<Plug-in id>_extn.yuix` extension file before extending or modifying forms, the Load/Create Extension File window

displays. You can either load an existing extension file or create and load the new extension file.

Table 6–18   Load/Create Extension File, Choose a File

Field	Description
Choose a File	
Loaded Extension Files	Select the extension file from the drop-down list, if applicable. If an extension file is not loaded in the tool, this combo box gets disabled.
click here	Click this hyperlink. The Choose File dialog displays. Select an extension file in which a new form element is automatically created. Click open. The extension file that you selected automatically gets loaded in the tool.
Load File	Click the Load File button to load the extension file that you selected in the Loaded Extension Files combo box.
Create New File	

Table 6–18   Load/Create Extension File, Choose a File

Field	Description
Plugin Id	Enter the plugin identifier of the plug-in that registers this new extension file.
	Press `Ctrl+Space` to select the plug-in Id from the list of available plug-ins.
Create and Load New File	When you click this button, the Save File dialog displays.
	If you are creating a new extension file, enter the extension file name and click Save. The new extension file gets created and loaded automatically. Otherwise, select an existing extension file in which a new form element is automatically created. Click Save. The extension file gets automatically loaded in the tool.

6.20.1.2 Viewing Screen Information

You can view the following information for a screen:

- The identifier of the editor contained by the screen.

- The identifier of the form is used to identify the screen.

- The identifiers of the task associated with the an editor.

- The `<Plug-in_id>_extn.yuix` extensions file associated to a particular screen.

- List of all the commands used by the screen to call APIs or services to get the required data for the screen.

- List of all the namespaces defined for the screen.

To view screen information:

1. Start the RCP Extensibility Tool. For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click i Screen Info . The Screen Information window displays.
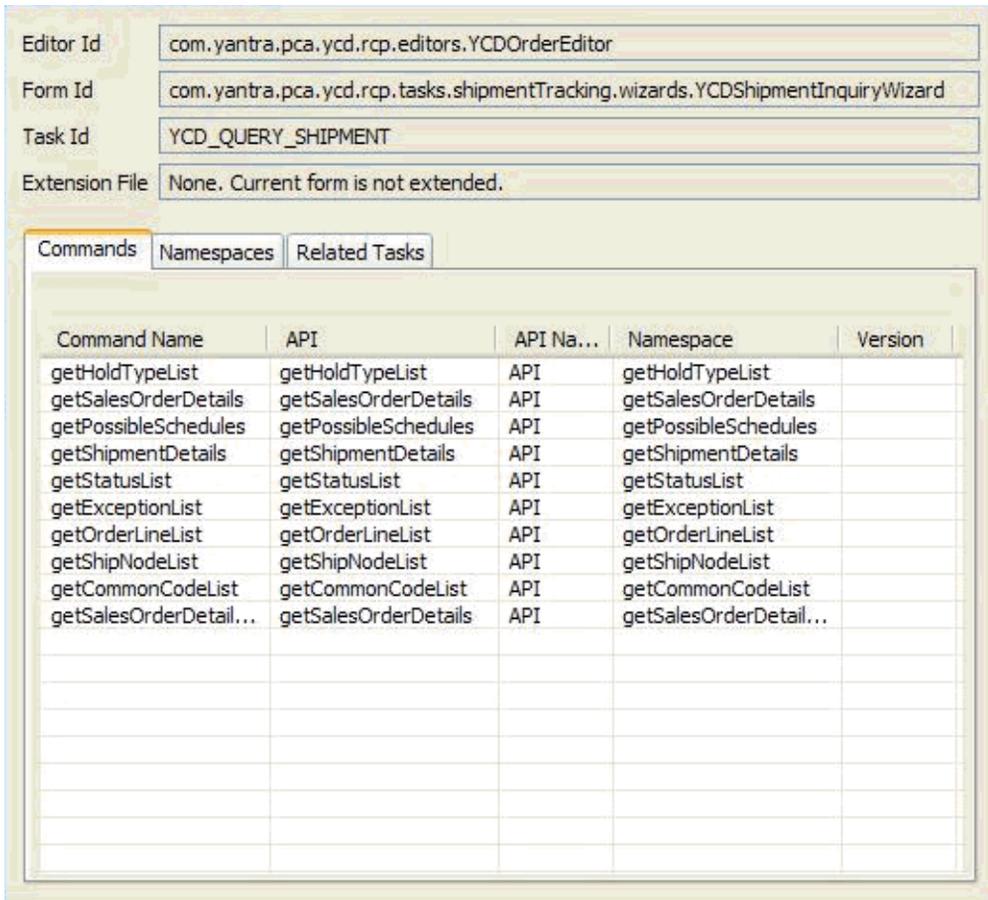
Table 6–19   Screen Information

Field	Description
Editor Id	The identifier of the editor contained by the screen.
Form Id	The identifier of the form is used to identify the screen.
Task Id	The identifiers of the task associated with the editor.
Extension File	The `<Plug-in_id>_extn.yuix` extensions file associated with the screen.

3. Select the commands tab to view all the commands that are being used by the screen.

4. Select the Namespaces tab to view the list of namespaces that are used by the screen. You can also view the XML model of the output template associated with the a namespace in the right hand side panel.

5. Select the Related Tasks tab to view the list of category id's in which this current active task is interested in. You can also view the list of group id's along with their group sequence number associated with the related tasks of the current active task.

6.20.1.3 Viewing Control Information

To view screen information:

1. Start the RCP Extensibility Tool. For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click **i Control Info** . The Control Information window displays. Figure displays the sample control info screen for a text box control.
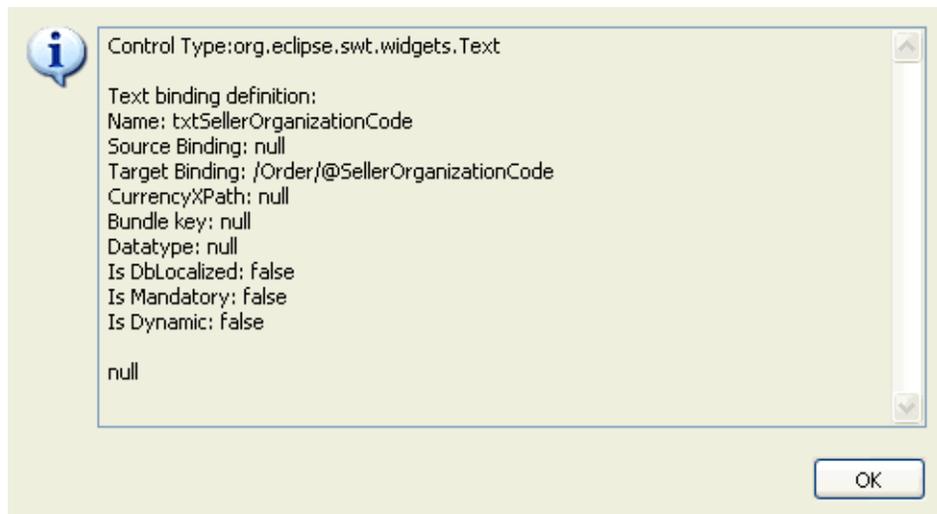
The system provides the following information for various controls:

- Control Type—The type of the control. For example, composite, group, button, and so forth.

- Name—The name of the control if applicable. For example, rdOpen, cmbPrice.

- Control Binding Definition—The information about the various bindings defined for a control. For example, source binding, target binding, checked binding.

  For more information about binding definitions for each control, see Section 6.8.5, "Binding Controls" Or see the *Sterling Supply Chain Applications Javadocs*.

6.20.1.4 Adding New Fields

You can add new fields to the screen and set properties such as layout, binding, theme, and so forth.

This section provides instructions for:

- Adding a Label

- Adding a Button

- Adding a Check Box

- Adding a Radio Button

- Adding a Text Box

- Adding a Styled Text

- Adding a Combo Box

- Adding a List Box

- Adding a Standard Or Advanced Table Column

- Adding a Link

- Adding a Composite

- Adding a Group

6.20.1.4.1 Adding a Label

To add a label:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click Label . Do one of the following to add a label:

- Select the control where you want to add the label, and click once.

- Select the composite or group where you want to add the label, and click once.

The Add Label window displays.

Table 6−20   Add Label

Field	Description
Control Name	Enter a unique control name for the new label. Every field on the form must have a unique logical name for reference. This field is mandatory. (Optional) Prefix the control name with "extn_". If you do not specify this, the system automatically adds "extn_" to the control name.

Table 6–20   Add Label

Field	Description
Default Value	Enter the default value to be displayed on the screen. This field is mandatory.
Attach	Select the place where you want to add the new label. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control". By default, the system displays "Attach within selected control" option only if you are adding the new label to an empty composite or group.

3. Click Finish to add the new label with the default layout data, theme and null binding attributes.

4. Click Next to specify the layout data options.

Table 6–21   Add Label, Layout Setup Page

Field	Description
Pick Layout Like	Select the type of layout associated with the control from the drop-down list.
	This combo box contains the list of control names that are siblings of the selected control. For more information about siblings, see Appendix B.10, "Siblings".
	To apply the same layout as one of the fields on the form, select that field from the combo box and view its layout in the form.
Pick This	When you click this button, all other fields get automatically populated based on the layout of the control you selected in the Pick Layout Like combo box.

Table 6–21   Add Label, Layout Setup Page

Field	Description
Horizontal Span	If you increase the horizontal span, the new field spans to the cell to the right of its current position.
Vertical Span	If you increase the Vertical span, the new field spans to the cell below its current position.
Grab Horizontal Space	Check this box if you want the new field to grab the extra horizontal space.
Grab Vertical Space	Check this box if you want the new field to grab the extra vertical space.
Horizontal Alignment	Select the appropriate horizontal alignment type for the new field. Valid values are 'BEGINNING', 'CENTER', 'END' or 'FILL'. BEGINNING aligns the new field to the left. CENTER aligns the new field to the center. END aligns the new field to the right. FILL fills the excess horizontal space.
Vertical Alignment	Select the appropriate vertical alignment type for the new field. Valid values are 'BEGINNING', 'CENTER', 'END' or 'FILL'. BEGINNING aligns the new field to the top. CENTER aligns the new field to the center. END aligns the new field to the bottom. FILL fills the excess vertical space.

5. Click Finish to add the new label with the default theme and null binding attributes.

6. Click Next to specify the binding options.

Table 6–22   Add Label, Binding Options Page

Source Binding	Enter the XML Path to populate the new field, if applicable.
	Click [...] to view the available XML Paths. The Source Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree.
	Press Ctrl+Space and select the appropriate XML Path from the drop-down list.

7. Click Finish to add the new label with the default theme binding.

8. Click Next to specify the theme binding.

Table 6–23   Add Label, Theme Binding Page

Field	Description
Theme Name	Enter the theme name you want to apply for the new field, if applicable.
	Click [...] to view the available themes. The Themes Tree pop-up window displays. Select the appropriate theme name you want to apply.
	Press Ctrl+Space and select the appropriate theme name you want to apply from the drop-down list.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, source binding, or theme entry for the new label. For more

information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.2 Adding a Button

To add a button:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click 🔲 Button . Do any of the following to add a button:

   - Select the control where you want to add the button and click once.
   - Select the composite or group where you want to add the button and click once.

   The Add Button window displays.

> **Note:** You can place the button either in the row layout
> container or grid layout container. However, the row layout
> container does not have layout data. Therefore, if you
> place the button in the row layout container, the Layout
> Setup Page does not display.

Control Name	extn_btnSearch
Default Value	Search
Attach	Attach after selected control ▼

☐ Validation Required?

Finish   Cancel

Table 6–24   Add Button

Field	Description
Control Name	Enter a unique control name for the new control. Every field on the form must have a unique logical name for reference.
	This field is mandatory.
	(Optional) Prefix the control name with "extn_". If you do not specify this, the system automatically adds "extn_" to the control name.
Default Value	Enter the default value to be displayed on the screen.
	This field is mandatory.
Attach	Select the place where you want to add the new control. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control".
	By default, the system displays "Attach within selected control" option only if you are adding the new control to an empty composite or group.
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.

3. Click Finish to add the new button with the default layout data and empty hot key binding.

4. Click Next to specify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new button with the empty hot key binding.

6. Click Next to specify the hot key binding.

Table 6–25 Add Button, Hot key Binding

Field	Description
Key Sequence	Enter a valid key sequence of the hot key for the control. For example, F7, M1+K. • Use M1 for specifying the Ctrl key. • Use M2 for specifying the Shift key. • Use M3 for specifying the Alt key. To specify a combination of keys use the "+" operator. For example, to specify the hot key for a control as "Ctrl+Alt+K, enter the key sequence as "M1+M3+K".
Action Definition	

Table 6–25   Add Button, Hot key Binding

Field	Description
Action Id	Enter the identifier of the action that you want to invoke when the hot key is pressed. For more information about action identifier and how to create actions, see Appendix B.6, "Creating New Actions". Note: The ActionId specified in this field should not exist in the `plugin.xml` file of your extension plug-in.
Action Class	Enter the fully qualified java classpath of the action class that you want to execute when the hot key is pressed. For more information about action class and how to create actions, see Appendix B.6, "Creating New Actions".

> **Note:** You cannot modify the hot key binding that is once entered and saved by you.

7. Click Finish.

8. Only if you specify the hot key binding, the "Restart Application" message box displays. The system prompts you to save the extension file and restart the application.

> **Note:** Whenever you specify the hot key binding for a control, you must save the extension file and restart the application to view the changes of the hot key binding. Make sure that you clear the configuration before starting the application. For more information about clearing configuration before launching an application, see Section 6.4.5, "Launching the Sterling Rich Client Application in Eclipse".

9. Click to save the changes made to the extension file.

10. Synchronize the resource files, if you have specified new default value for the new button. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

11. (Optional) If you want to implement the logic on button click event do the following:

> **Note:** While adding the new button, make sure that you check the "Validation Required?" box.

   a. Synchronize the extension behavior for the button. For more information about synchronizing extension behavior, see Section 6.20.1.8.4, "Synchronizing Extension Behavior".

   b. Start Eclipse SDK.

   c. In the navigator view, expand the plug-in project, which contains this screen.

   d. Expand the package and open the extension behavior class, which you specified while synchronizing extension behavior. For more information about synchronizing extension behavior, see Section 6.20.1.8.4, "Synchronizing Extension Behavior".

   e. In the validateButtonClick() method, add the logic to provide the implementation for the button click event.

6.20.1.4.3 Adding a Check Box

To add a check box:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ☑ Check Box . Do any of the following to add a check box:

- Select the control where you want to add the check box and click once.

- Select the composite or group where you want to add the check box and click once.

The Add Check Box window displays. For field value descriptions, see Table 6–24.

> **Note:** You can place the check box either in the row layout container or grid layout container. However, the row layout container does not have layout data. Therefore, if you place the check box in the row layout container, the Layout Setup Page does not display.

3. Click Finish to add the new check box with the default layout data, theme and null binding attributes.

4. Click Next to specify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new check box with the default theme and null binding attributes.

6. Click Next to specify the binding options.

Data Binding

Source Binding	/Order/@IsAccrossEnterprice	...
Target Binding	/Order/@IsAccrossEnterprice	...
Checked Binding	Y	...
Unchecked Binding	N	...

< Back   Next >   Finish   Cancel

Table 6–26   Add Check Box, Binding Options Page

Field	Description
Source Binding	Enter the XML Path to populate the new field, if applicable.
	Click […] to view the available XML Paths. The Source Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree.
	Press Ctrl+Space and select the appropriate XML Path from the drop-down list.
Target Binding	Enter the XML Path to send data to the API for the new field, if applicable. Multiple target bindings can be specified using a semi-colon.
	Click […] to view the available XML Paths. The Target Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree.
	Press Ctrl+Space and select the appropriate XML Path from the drop-down list.
Checked Binding	Enter the XML Path to specify the checked binding attribute, if applicable.
	Click […] to view the available XML Paths. The Checked Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree.
	Press Ctrl+Space and select the appropriate XML Path from the drop-down list.
Unchecked Binding	Enter the XML Path to specify the unchecked binding attribute, if applicable.
	Click […] to view the available XML Paths. The Unchecked Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree.
	Press Ctrl+Space and select the appropriate XML Path from the drop-down list.

7. Click Finish to add the new check box with the default theme binding.

8. Click Next to specify theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click [icon] to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the new check box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.4   Adding a Radio Button

To add a radio button:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click [⊙ Radio Button] . Do any of the following to add a radio button:

- Select the control where you want to add the radio button and click once.

- Select the composite or group where you want to add the radio button and click once.

The Add Radio Button window displays. For field value descriptions, see Table 6–24.

> **Note:**   You can place the radio button either in the row layout container or grid layout container. However, the row layout container does not have layout data. Therefore, if you place the radio button in the row layout container, the Layout Setup Page does not display.

3. Click Finish to add the new radio button with the default layout data, theme and null binding attributes.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new radio button with the default theme and null binding attributes.

6. Click Next to specify the binding options.

For field value descriptions, see Table 6–26.

7. Click Finish to add the new radio button with the default theme binding.

8. Click Next to specify theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the new radio button. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.5 Adding a Text Box

To add a text box:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ⬚ Text Box . Do any of the following to add a text box:

   • Select the control where you want to add the text box and click once.

   • Select the composite or group where you want to add the text box and click once.

   The Add Text Box window displays. For field value descriptions, see Table 6–24.

3. Click Finish to add the new text box with the default layout data, theme and null binding attributes.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new text box with the default theme and null binding attributes.

6. Click Next to specify the binding options.

For field value descriptions, see Table 6–26.

7. Click Finish to add the new text box with the default theme binding.

8. Click Next to specify theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the new text box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.6 Adding a Styled Text

The StyledText component is a customized control that can be used to display and edit text with different colors and font styles. The StyledText components are powerful and multifaceted components suitable for high-end needs, and offer more avenues for customization than the other text components.

To add a styled text:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click StyledText . Do any of the following to add a styled text:

   - Select the control where you want to add the styled text and click once.

   - Select the composite or group where you want to add the styled text and click once.

   The Add Styled Text window displays. For field value descriptions, see Table 6–24.

3. Click Finish to add the new styled text with the default layout data, theme and null binding attributes.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new styled text with the default theme and null binding attributes.

6. Click Next to specify the binding options.

For field value descriptions, see Table 6–26.

7. Click Finish to add the new styled text with the default theme binding.

8. Click Next to specify theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 📋 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the new styled text. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.7 Adding a Combo Box

To add a combo box:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click Combo Box . Do any of the following to add a combo box:

   • Select the control where you want to add the combo box and click once.

   • Select the composite or group where you want to add the combo box and click once.

   The Add Combo Box window displays.

Table 6–27   Add Combo Box

Field	Description
Control Name	Enter a unique control name for the new control. Every field on the form must have a unique logical name for reference. This field is mandatory. (Optional) Prefix the control name with "extn_". If you do not specify this, the system automatically adds "extn_" to the control name.
Default Value	Enter the default value to be displayed on the screen. This field is mandatory.

Table 6–27   Add Combo Box

Field	Description
Attach	Select the place where you want to add the new control. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control". By default, the system displays "Attach within selected control" option only if you are adding the new control to an empty composite or group.
DB Localized?	Check this box, to retrieve data for the new field from a localized database (DB) column, if applicable.
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.

3. Click Finish to add the new combo box with the default layout data, theme and null binding attributes.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new combo box with the default theme and null binding attributes.

6. Check this box, to retrieve data for the new field from a localized database (DB) column, if applicable.

7. Click Next to specify the binding options.

Table 6–28   Add Combo Box, Binding Options Page, Data Binding

Field	Description
Source Binding	Enter the XML Path to populate the new field, if applicable. Click `…` to view the available XML Paths. The Source Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree. Press Ctrl+Space and select the appropriate XML Path from the drop-down list.
Target Binding	Enter the XML Path to send data to the API for the new field, if applicable. Multiple target bindings can be specified using a semi-colon. Click `…` to view the available XML Paths. The Target Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree. Press Ctrl+Space and select the appropriate XML Path from the drop-down list.
List Binding	Enter the XML Path to get the list of items for the new field. Click `…` to view the available XML Paths. The List Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree. Press Ctrl+Space and select the appropriate XML Path from the drop-down list.

Table 6−28   Add Combo Box, Binding Options Page, Data Binding

Field	Description
Code Binding	Enter the XML Path to send the value of an attribute to the API for the new field. Click ⌷⋯⌷ to view the available XML Paths. The Code Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree. Press Ctrl+Space and select the appropriate XML Path from the drop-down list.
Description Binding	Enter the XML Path to specify the value to be displayed on the screen. Click ⌷⋯⌷ to view the available XML Paths. The Description Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree. Press Ctrl+Space and select the appropriate XML Path from the drop-down list.

8. Click Finish to add the new combo box with the default theme binding.

9. Click Next to specify theme binding. For field value descriptions, see Table 6−23.

10. Click Finish.

11. Click on ⎙ save the changes made to the extension file.

12. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the new combo box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.8   Adding a List Box

To add a list box:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ▦ List Box . Do any of the following to add a list box:

- Select the control where you want to add the list box and click once.

- Select the composite or group where you want to add the list box and click once.

The Add List Box window displays.

For field value descriptions, see Table 6–27.

3. Click Finish to add the new list box with the default layout data, theme and null binding attributes.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new list box with the default theme and null binding attributes.

6. Click Next to specify the binding options. For field value descriptions, see Table 6–28.

7. Click Finish to add the new list box with the default theme binding.

8. Click Next to specify theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new bindings, or theme entry for the new list box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.9   Adding a Standard Or Advanced Table Column

You can either add a standard column or advanced column to the table.

To add a standard or advanced column to the table:

1. Start the RCP Extensibility Tool.

    For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ▊ Table Column . Select the column where you want to add the new standard or advanced column and click once. The Add Table Column window displays.

Table 6–29   Add Table Column

Field	Description
Control Name	Enter a unique control name for the new column. Every field on the form must have a unique logical name for reference.
	This field is mandatory.
	(Optional) Prefix the control name with "extn_". If you do not specify this, the system automatically adds "extn_" to the control name.
Default Value	Enter the default value to display on the screen.
	This field is mandatory.

Table 6—29   Add Table Column

Field	Description
Attach	Place the cursor on the screen where you want to add the new control. Valid values are "Attach after selected control" and "Attach before selected control".
DB Localized?	Check this box, if you want to retrieve data for this new column from a localized DB column.
Movable?	Check this box to move the column, if applicable.
Sort Required?	Check this box to sort data in a particular order, if applicable.
Filter Required?	Check this box to filter data, if applicable.
Standard Column	Choose this if you want to add a standard column.
Advanced Column	Choose this if you want to add an advanced column.

3. Click Finish to add the new column with the default column layout data, theme, and null binding attributes.

4. Click Next to specify the layout data options.

Table 6–30   Add Table Column, Layout Setup Page

Field	Description
Column Width	Increase or decrease the width of the new column, if necessary.

5. Click Finish to add the new column with the default theme and null binding attributes.

6. Click Next to specify the binding options.

Table 6–31   Add Table Column, Binding Options Page

Field	Description
Attribute Binding	Enter the XML Path to specify the binding attribute, if applicable. Click ⌷ to view the available XML Paths. The Attribute Bindings Tree pop-up window displays. Select the appropriate XML Path from the tree. Press Ctrl+Space and select the appropriate XML Path from the drop-down list. NOTE: If you are adding an advanced column, this field is disabled. You cannot specify the attribute binding for an advanced column using the Sterling RCP Extensibility Tool.

7. Click Finish to add the new column with the default theme binding.

8. Click Next to specify the theme binding.

Table 6–32   Add Table Column,

Field	Description
Theme Name	Enter the theme name you want to apply for the new field, if applicable.
	Click ⬚ to view the available themes. The Themes Tree pop-up window displays. Select the appropriate theme name you want to apply.
	Press Ctrl+Space and select the appropriate theme name you want to apply from the drop-down list.
	NOTE: If you are adding an advanced column, this field is disabled. You cannot specify the theme binding for an advanced column using the Sterling RCP Extensibility Tool.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, attribute binding, or theme entry for the new column. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

> **Note:**   If you have added an advanced column, create and synchronize the appropriate extension behavior for setting the bindings for the extended table. For more information about creating extension behavior, see Section 6.20.1.5, "Creating Extension Behavior". For more information about synchronizing extension behavior, see Section 6.20.1.8.4, "Synchronizing Extension Behavior".
>
> After creating and synchronizing the extension behavior, set the bindings for the extended table and advanced column that you added in the extension behavior class. For more information about binding extended tables, see Binding Extended Tables.

6.20.1.4.10 Adding a Link

To add a link:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ![Link icon] Link . Do any of the following to add a link:

   • Select the control where you want to add the link and click once.

   • Select the composite or group where you want to add the link and click once.

   The Add Link window displays. For field value descriptions, see Table 6–24.

   > **Note:** You can place the link either in the row layout container or grid layout container. However, the row layout container does not have layout data. Therefore, if you place the link in the row layout container, the Layout Setup Page does not display.

3. Click Finish to add the new link with the default layout data, theme null binding attributes, and empty hot key binding.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–21.

5. Click Finish to add the new link with the default theme, null binding attributes, and empty hot key binding.

6. Click Next to specify the binding options.

   For field value descriptions, see Table 6–24.

Data Binding

Source Binding | ItemDetails:Item/@ItemID | [...]

Target Binding | ItemDetails:Item/@ShortDescription | [...]

[< Back] [Next >] [Finish] [Cancel]

7. Click Finish to add the new link with the default theme binding and empty hot key binding.

8. Click Next to specify theme binding. For field value descriptions, see Table 6–23.

9. Click Finish to add the new link with empty hot key binding.

10. Click Next to specify hot key binding. For field value descriptions, see Table 6–25.

11. Click Finish.

12. Only if you specify the hot key binding, the "Restart Application" message box displays. The system prompts you to save the extension file and restart the application.

> **Note:** Whenever you specify the hot key binding for a control, you must save the extension file and restart the application to view the changes of the hot key binding. Make sure that you clear the configuration before starting the application. For more information on about clearing configuration before launching an application, see Section 6.4.5, "Launching the Sterling Rich Client Application in Eclipse".

13. Click ▦ to save the changes made to the extension file.

14. Synchronize the resource files, if you have specified new default values, bindings, or theme entries for the new link. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.4.11   Adding a Composite

To add a composite:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ☐ Composite . Do any of the following to add a composite:

- Select the control where you want to add the composite and click once.

- Select the composite or group where you want to add the composite and click once.

The Add Composite window displays. For field value descriptions, see Table 6–20.

3. Click Finish to add the new composite with the default layout data.

4. Click Next to specify layout data options.

Table 6–33   Add Composite, Layout Setup Page, Options

Field	Description
Layout Data Options	
Pick Layout Like	Select the type of layout associated with the control from the drop-down list.
	This combo box contains the list of control names that are siblings of the selected control. For more information about siblings, see Appendix B.10, "Siblings".
	To apply the same layout as one of the fields on the form, select that field from the combo box and view its layout in the form.
Pick This	When you click this button, all other fields get automatically populated based on the layout of the control you selected in the Pick Layout Like combo box.

Table 6–33   Add Composite, Layout Setup Page, Options

Field	Description
Horizontal Span	If you increase the horizontal span, the new field spans to the cell to the right of its current position.
Vertical Span	If you increase the Vertical span, the new field spans to the cell below its current position.
Grab Horizontal Space	Check this box if you want the new field to grab the extra horizontal space.
Grab Vertical Space	Check this box if you want the new field to grab the extra vertical space.
Horizontal Alignment	Select the appropriate horizontal alignment type for the new field. Valid values are 'BEGINNING', 'CENTER', 'END' or 'FILL'.
	BEGINNING aligns the new field to the left.
	CENTER aligns the new field to the center.
	END aligns the new field to the right.
	FILL fills the excess horizontal space.
Vertical Alignment	Select the appropriate vertical alignment type for the new field. Valid values are 'BEGINNING', 'CENTER', 'END' or 'FILL'.
	BEGINNING aligns the new field to the top.
	CENTER aligns the new field to the center.
	END aligns the new field to the bottom.
	FILL fills the excess vertical space.
Layout Options	
Layout	Select the layout you want to apply for the new composite.
Number of Columns	Enter or click ⬍ to increase or decrease the number of columns, if applicable.
Make Columns Equal	Check this box to make all columns of equal width for this new composite, if applicable.

5. Enter the required values and click Finish.

6. Click 🖫 to save the changes made to the extension file.

6.20.1.4.12 Adding a Group

To add a group:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click Group . Do any of the following to add a group:

- Select the control where you want to add the group and click once.

- Select the composite or group where you want to add the group and click once.

The Add Group window displays. For field value descriptions, see Table 6–20.

3. Click Finish to add the new group with the default layout data.

4. Click Next to specify layout data options. For field value descriptions, see Table 6–33.

5. Click Finish.

6. Click to save the changes made to the extension file.

7. Synchronize the resource files, if you have specified new default value for the new group. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.5 Creating Extension Behavior

You must create the extension behavior whenever you are extending an existing wizard or screen:

- To perform custom validations for the newly added fields

- To call an API or Sevice

- To set the extension model to populate a screen or wizard page

- To set bindings for advanced columns that you add through the Sterling RCP Extensibility Tool.

If you are extending a screen which is not a wizard, the tool creates the normal extension behavior and if you are extending a screen which is a wizard, the tool creates the wizard extension behavior.

To create extension behavior:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click = Extension Behavior and click on the form. The Create Extension Behavior? message displays. The message box prompts you to confirm whether you want to create an extension behavior for the current form. The message box also displays the identifier of the current form.

3. Click OK.

4. Synchronize the extension behavior that you created to specify the implementation class, if applicable. For more information about synchronizing extension behavior, see Section 6.20.1.8.4, "Synchronizing Extension Behavior".

6.20.1.6 Modifying or Deleting the Newly Added Fields

You can modify the newly added fields such as disabling, hiding, changing their default values, and so forth. The ability to modify layout settings, binding attributes, and theme is also provided. You can also delete the newly added fields from the screen.

> **Note:** You cannot modify the control name of any field.

6.20.1.6.1 Modifying a Newly Added Label

To modify a newly added label:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added label and click once. The Edit New Label window displays.

Table 6–34   Edit New Label

Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference. This field is disabled.
Default Value	Change the default value to be displayed on the screen, if applicable.

Table 6–34   Edit New Label

Field	Description
Attach	Select the place where you want to place the field. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control". By default, the system displays "Attach within selected control" option only if you are editing a field, which is placed in a composite or group.
Delete	Click this button to delete the newly added field from the form and also from the extension file.

3. Click Finish. The newly added label updates with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added label is updated with existing binding attributes and theme.

6. Click Next to modify the binding options. For field value descriptions, see Table 6–22.

7. Click Finish. The newly added label is updated with the existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click ▣ to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, source binding, or theme entry for the newly added label. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.2 Modifying a Newly Added Button

To modify a newly added button:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select the newly added button and click once. The Edit New Button window displays.

Note: If you are modifying a button that is placed in the row layout container, the Layout Setup Page window does not display as the layout data is not available.

Table 6–35   Edit New Button

Field	Description
Control Name	The control name of any field on the form is a unique logical name for reference.
	This field is disabled.
Default Value	Change the default value to be displayed on the screen, if applicable.

Table 6–35   Edit New Button

Field	Description
Attach	Place the cursor on the screen where you want to place the field. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control". By default, the system displays "Attach within selected control" option only if you are editing a field, which is placed in a composite or group.
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.
Delete	Click this button to delete the newly added field from the form and also from the extension file.

3. Click Finish. The newly added button is updated with the existing layout data.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish.

6. Click ⬚ to save the changes made to the extension file.

7. Synchronize the resource files, if you have specified new default value for the newly added button. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.3   Modifying a Newly Added Check Box

To modify a newly added check box:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select the newly added check box and click once. The Edit New Check Box window displays. For field value descriptions, see Table 6–35.

> **Note:** If you are modifying a check box that is placed in
> the row layout container, the Layout Setup Page window
> does not display. This is because the row layout does not
> have the layout data.

3. Click Finish. The newly added check box is updated with existing
   layout data, binding attributes, and theme.

4. Click Next to modify layout data options. For field value descriptions,
   see Table 6–21.

5. Click Finish. The newly added check box is updated with existing
   binding attributes and theme.

6. Click Next to modify the binding options. For field value descriptions,
   see Table 6–26.

7. Click Finish. The newly added check box is updated with the existing
   theme.

8. Click Next to modify the theme binding. For field value descriptions,
   see Table 6–23.

9. Click Finish.

10. Click 📇 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default
    value, bindings, or theme entry for the newly added check box. For
    more information about synchronizing resource files, see
    Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.4 Modifying a Newly Added Radio Button

To modify a newly added radio button:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1,
   "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added radio button and click once. The Edit New Radio
   Button window displays. For field value descriptions, see Table 6–35.

> **Note:** If you are modifying a radio button that is placed in the row layout container, the Layout Setup Page window does not display. This is because the row layout does not have the layout data.

3. Click Finish. The newly added radio button is updated with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added radio button is updated with existing binding attributes and theme.

6. Click Next to modify the binding options.

For field value descriptions, see Table 6–26.

7. Click Finish. The newly added radio button is updated with the existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click ▦ to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the newly added radio button. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.5 Modifying a Newly Added Text Box

To modify a newly added text box:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added text box and click once. The Edit New Text Box window displays.

Table 6–36   Edit New Text Box

Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference.
	This field is disabled.
Default Value	Change the default value to be displayed on the screen, if applicable.
Attach	Select the place where you want to place the field. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control".
	By default, the system displays "Attach within selected control" option only if you are editing a field, which is placed in a composite or group.
DB Localized?	Check this box, to retrieve data for the new field from a localized database (DB) column, if applicable.

Table 6–36   Edit New Text Box

Field	Description
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.
Delete	Click this button to delete the newly added field from the form and also from the extension file.

3. Click Finish. The newly added text box is updated with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added text box is updated with existing binding attributes and theme.

6. Click Next to modify the binding options.

For field value descriptions, see Table 6–26.

7. Click Finish. The newly added text box is edited with the existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 🔳 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the newly added text box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.6 Modifying a New Added Styled Text

To modify a newly added styled text:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added styled text and click once. The Edit New Styled Text window displays. For field value descriptions, see Table 6–36.

3. Click Finish. The newly added styled text is updated with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added styled text is updated with existing binding attributes and theme.

6. Click Next to modify the binding options.

For field value descriptions, see Table 6–26.

7. Click Finish. The newly added styled text is updated with the existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the newly added styled text. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.7 Modifying a Newly Added Combo Box

To modify a newly added combo box:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added combo box and click once. The Edit New Combo Box window displays. For field value descriptions, see Table 6–36.

3. Click Finish. The newly added combo box is updated with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added combo box is updated with existing binding attributes and theme.

6. Click Next to modify the binding options. For field value descriptions, see Table 6–28.

7. Click Finish. The newly added combo box is updated with existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the newly added combo box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.8 Modifying a Newly Added List Box

To modify a newly added list box:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added list box and click once. The Edit New List Box window displays. For field value descriptions, see Table 6–36.

For field value descriptions, see Table 6–36.

3. Click Finish. The newly added list box is updated with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added list box is updated with existing binding attributes and theme.

6. Click Next to modify the binding options. For field value descriptions, see Table 6–28.

7. Click Finish. The newly added list box is updated with the existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new bindings, or theme entry for the newly added list box. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.9 Modifying a Newly Added Table Column

To modify a newly added table column:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added column in a table and click once. The Edit New Table Column window displays.

Table 6−37   Edit New Table Column

Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference. This field is disabled.
Default Value	Change the default value to be displayed on the screen, if applicable.
Attach	Select the place where you want to place the field. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control". By default, the system displays "Attach within selected control" option only if you are editing a field, which is placed in a composite or group.

Table 6–37   Edit New Table Column

Field	Description
DB Localized?	Check this box, to retrieve data for the new field from a localized database (DB) column, if applicable.
Sort Required?	Check this box to sort the data in a particular order, if applicable.
Filter Required?	Check this box to filter the data, if applicable.
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.
Movable?	Check this box to move the table columns, if applicable.
Delete	Click this button to delete the newly added field from the form and also from the extension file.

3. Click Finish. The newly added table column is updated with existing binding attributes and theme.

4. Click Next to modify the binding options. For field value descriptions, see Table 6–31.

5. Click Finish. The newly added table column is updated with the existing theme.

6. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

7. Click Finish.

8. Click 🖫 to save the changes made to the extension file.

9. Synchronize the resource files, if you have specified new default value, attribute binding, or theme entry for the newly added table column. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.10 Modifying a Newly Added Link

To modify a newly added link:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added link and click once. The Edit New Link window displays.

> **Note:** If you are modifying a link that is placed in the row layout container, the Layout Setup Page window does not display. This is because the row layout does not have the layout data.

For field value descriptions, see Table 6–36.

3. Click Finish. The newly added link is updated with existing layout data, binding attributes, and theme.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–21.

5. Click Finish. The newly added link is updated with existing binding attributes and theme.

6. Click Next to modify the binding options.

For field value descriptions, see Table 6–28.

7. Click Finish. The newly added link is updated with the existing theme.

8. Click Next to modify the theme binding. For field value descriptions, see Table 6–23.

9. Click Finish.

10. Click 🖫 to save the changes made to the extension file.

11. Synchronize the resource files, if you have specified new default value, bindings, or theme entry for the newly added link. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.6.11 Modifying a Newly Added Composite

To modify a newly added composite:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added composite and click once. The Edit New Composite window displays.

Table 6–38   Edit New Composite

Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference.
	This field is disabled.
Attach	Select the place where you want to place the field. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control".
	By default, the system displays "Attach within selected control" option only if you are editing a field, which is placed in a composite or group.

Table 6–38   Edit New Composite

Field	Description
Delete	Click this button to delete the newly added field from the form and also from the extension file.
Delete Cascade Within	Click this button to delete the newly added field and its children controls from the form and also from the extension file.

3. Click Finish. The newly added composite is edited with the existing layout data.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–33.

5. Click Finish.

6. Click 🖫 to save the changes made to the extension file.

6.20.1.6.12   Modifying a Newly Added Group

To modify a newly added group:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Select a newly added group and click once. The Edit New Group window displays.

Table 6–39   Edit New Group

Field	Description
Control Name?	Control Name of any field on the form is a unique logical name for reference. This field is disabled.
Default Value	Change the default value to be displayed on the screen, if applicable.
Attach	Select the place where you want to place the field. Valid values are "Attach after selected control", "Attach before selected control", and "Attach within selected control". By default, the system displays "Attach within selected control" option only if you are editing a field, which is placed in a composite or group.

Table 6–39   Edit New Group

Field	Description
Delete	Click this button to delete the newly added field from the form and also from the extension file.
Delete Cascade Within	Click this button to delete the newly added field and its children controls from the form and also from the extension file.

3. Click Finish. The newly added group is updated with the existing layout data.

4. Click Next to modify the layout data options. For field value descriptions, see Table 6–33.

5. Click Finish.

6. Click 🖫 to save the changes made to the extension file.

7. Synchronize the resource files, if you have specified new default value for the newly added group. For more information about synchronizing resource files, see Section 6.20.1.8, "Synchronizing Resource Files".

6.20.1.7 Modifying Existing Fields

You can modify the Sterling RCP-provided existing fields such as disabling, hiding, changing their default values, and so forth. However, Sterling RCP does not allow you to modify existing composites and groups.

> **Note:**   You cannot delete any existing fields from the form, or modify the control name of any existing fields.

6.20.1.7.1   Modifying an Existing Label or Combo Box

To modify an existing label or combo box:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. (Optional) To show hidden and disabled fields, click ▭.

3. Select an existing field and click once. The Edit Existing Field window displays.

Table 6–40 Edit Existing Field

Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference. This field is disabled.
Enabled?	Uncheck this box, if you want to disable the field.

Table 6–40   Edit Existing Field

Field	Description
Visible?	Uncheck this box, if you want to hide the field.
Delete	This button appears only when you make modifications to an existing field.
	When you click this button, the field gets deleted from the extension file. Also, all modifications made to this field gets deleted, and the default settings for this field are automatically restored.

4. Click Finish.

5. Click ▦ to save the changes made to the extension file.

6.20.1.7.2   Modifying an Existing Button, Check Box, Radio Button, List Box, or Link.

To modify an existing button, check box, radio button, list box, or link:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. (Optional) To show hidden and disabled fields, click ▭.

3. Select an existing field and click once. The Edit Existing Field window displays.

Table 6–41   Edit Existing Field

Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference. This field is disabled.
Enabled?	Uncheck this box, if you want to disable the field.
Visible?	Uncheck this box, if you want to hide the field.

Table 6–41   Edit Existing Field

Field	Description
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.
Delete	This button appears only when you make modifications to an existing field.
	When you click this button, the field gets deleted from the extension file. Also, all modifications made to this field gets deleted, and the default settings for this field are automatically restored.

4. Click Finish.

5. Click ▨ to save the changes made to the extension file.

6.20.1.7.3   Modifying an Existing Text Box or Styled Text To modify an existing text box or styled text:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. (Optional) To show hidden and disabled fields, click ▭.

3. Select an existing field and click once. The Edit Existing Field window displays.

Table 6–42   Edit Existing Field

| Field	Description
Control Name	Control Name of any field on the form is a unique logical name for reference. This field is disabled.
Default Value	Change the default value to be displayed on the screen, if applicable.
Enabled?	Uncheck this box, if you want to disable the field.
Visible?	Uncheck this box, if you want to hide the field.

Table 6–42   Edit Existing Field

Field	Description
Validation Required?	Check this box to validate the new field, if applicable. Sterling RCP invokes the validation method on the extended behavior class of this form when the field looses focus.
Delete	This button appears only when you make modifications to an existing field.
	When you click this button, the field gets deleted from the extension file. Also, all modifications made to this field gets deleted, and the default settings for this field are automatically restored.

4. Click Finish.

5. Click to save the changes made to the extension file.

6.20.1.7.4 Modifying an Existing Table Column To modify an existing table column:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. (Optional) To show hidden and disabled fields, click .

3. Select an existing column in the table and click once. The Edit Existing Field window displays.
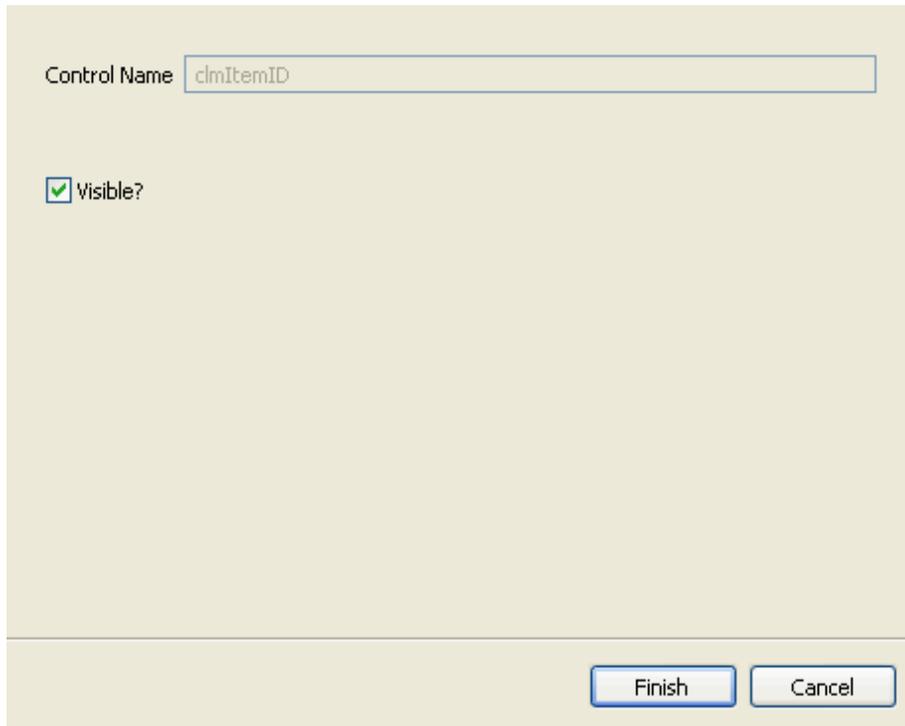
For field value descriptions, see Table 6–40.

4. Click Finish.

5. Click 🖫 to save the changes made to the extension file.

6.20.1.8 Synchronizing Resource Files
Whenever you add or modify fields, you must perform the following:

- Synchronizing Bundle Entries

- Synchronizing Templates

- Synchronizing Theme Files

- Synchronizing Extension Behavior

6.20.1.8.1 Synchronizing Bundle Entries

Whenever you add new fields, you must synchronize the existing bundle file with new bundle entries.

To synchronize the bundle file:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click 📄. The Synchronize pop-up window displays.

Bundle Entries	Template	Theme	Extn Behavior

Key	Value
Order_No	OrderNo
Reset	Reset
Include_All_Items	Include All Required Items
Credit_Card	Credit Card

Save All   Close
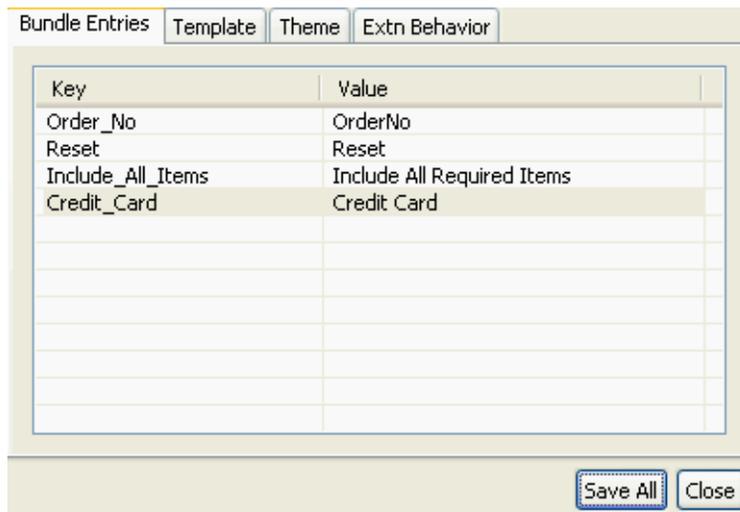
3. Select the Bundle Entries tab.

4. In the key column, you can view the control names of all the newly added fields for which the corresponding bundle entry does not exist.

5. In the Value column, enter the value associated with the key, if applicable. This value is used when retrieving the value for the keys from the bundle file.

6. Click Save All. The Save All window displays.

Table 6–43   Choose File for Saving Diffs

Field	Description
Bundle File	Click Browse. The Save File dialog box displays. Select the bundle file in which you want to save the new bundle entries.
Theme File	Click Browse. The Save File dialog box displays. Select the theme file in which you want to save the new theme entries.
Template Folder	Click Browse. The Choose Folder dialog box displays. Select the template folder in which you want to save the new template.
Extn Behavior Folder	Click Browse. The Choose Folder dialog box displays. Select the folder in which you want to save the new extension behavior class.

> **Note:** Sterling Commerce recommends that you save all files, folders, and packages in the plug-in directory.

7. Click Save.

8. Click Close.

6.20.1.8.2 Synchronizing Templates

Whenever you specify bindings for the newly added fields, you must synchronize the template folder with new templates that are defined for binding the newly added fields.

To synchronize the templates:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click . The Synchronize pop-up window displays. For more information, see Figure .

3. Select the Template tab. You can view the newly added bindings for the newly added fields under the respective form id.

4. Click Save All. The Choose File for Saving Diffs pop-up window displays. For field value descriptions, see Table 6–43.

5. Click Close.

6.20.1.8.3 Synchronizing Theme Files

Whenever you specify a new theme entry for the newly added fields, you must synchronize the theme file.

To synchronize the theme file:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click . The Synchronize pop-up window displays. For more information, see Figure .

3. Select the Theme tab. In the Themes column, you can view the new theme entries.

4. Click Save All. The Choose File for Saving Diffs pop-up window displays. For field value descriptions, see Table 6–43.

5. Click Close.

6.20.1.8.4 Synchronizing Extension Behavior

You must synchronize the extension behaviors whenever you create it.

For more information about creating extension behavior, see Section 6.20.1.5, "Creating Extension Behavior".

There are two types of extension behavior:

- Normal Extension Behavior (YRCExtensionBehavior)—The system creates this behavior when you create an extension behavior for extending a Sterling Rich Client screen that is not a wizard. When synchronizing a normal extension behavior, specify the name of the implementation class (extends YRCExtensionBehavior). You can add the logic for calling an API or service or setting the extension model in the init() method of the YRCExtensionBehavior class.

- Wizard Extension Behavior (YRCWizardExtensionBehavior)—The system creates this behavior when you create the extension behavior for extending a Sterling Rich Client screen that is a wizard. When synchronizing a wizard extension behavior, specify the name of the implementation class (extends YRCWizardExtensionBehavior). You can add the wizard specific logic for calling an API OR service or setting the extension model in the initPage (String pageId) method of the YRCWizardExtensionBehavior class.

To synchronize the extension behavior:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ⧉. The Synchronize pop-up window displays. For more information, see Figure .

3. Select the Extn Behavior tab.

4. In FormId column, you can view the identifier of the form that you are extending.

5. In Behavior column, the type of behavior class automatically displays in the combo box. For example, WizardExtnBehavior or ExtnBehavior.

   - If you have created the extension behavior for a Sterling Rich Client screen, which is a wizard, the combo box displays "WizardExtnBehavior".

   - If you have created the extension behavior for a Sterling Rich Client screen, which is not a wizard, the combo box displays "ExtnBehavior".

6. In ClassName column, enter the path of the behavior class. For example:

   ```
   com.yantra.pca.ycd.ui.screens.MyBehaviorClass
   ```

   where `com.yantra.pca.ycd.ui.screens` is the package name and `MyBehaviorClass` is the name of the behavior class.

7. Click Save All. The Choose File for Saving Diffs pop-up window displays. For field value descriptions, see Table 6–43.

8. Click Close. The extension behavior class is created in the package that you specified.

9. To retrieve the appropriate data for this newly added field or advanced column, write the code for calling the required API or service, setting the extension model, and setting the bindings for advanced column in the newly created extension behavior class.

   Calling APIs or Services

   After synchronizing the extension behavior for a newly added field, you may want to retrieve data to populate the new field by calling an API or service. For more information about calling APIs or services, see Section 6.8.8, "Calling APIs and Services".

   Setting the Extension Model

   You must set the extension model for the new namespaces that you define for the extended screens. For more information about extension model, see Section 6.19, "Setting the Extension Model".

   Setting Bindings for Advanced Column

   You must set bindings for the extended table and advanced column that you added using the Sterling RCP Extensibility Tool. For more

information about binding the extended table and advanced column, see Binding Extended Tables.

6.20.1.9 Showing Hidden and Disabled Fields

You can show the hidden and disabled fields that are provided with Sterling Supply Chain Applications using the RCP Extensibility Tool.

To show the hidden and disabled fields:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click ▭. The hidden and disabled fields display on the screen.

6.20.1.10 Adding Secure APIs

Using the RCP Extensibility Tool, in the `secureapis.xml` file, you can add certain sensitive APIs that you want to call by making an SSL call instead of a normal HTTP call.

You can make SSL calls to the APIs at the following levels:

In the `secureapis.xml` file, you can either directly add the name of the APIs that you want to call securely at the API level, or search and add the names of the commands that you want to call securely at the command level. You can search for such commands by passing the attributes of an API in the search criteria.

To add the secure APIs:

1. Start the RCP Extensibility Tool.

For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click Add Secure APIs and click once on the screen. The Add Secure APIs window displays.
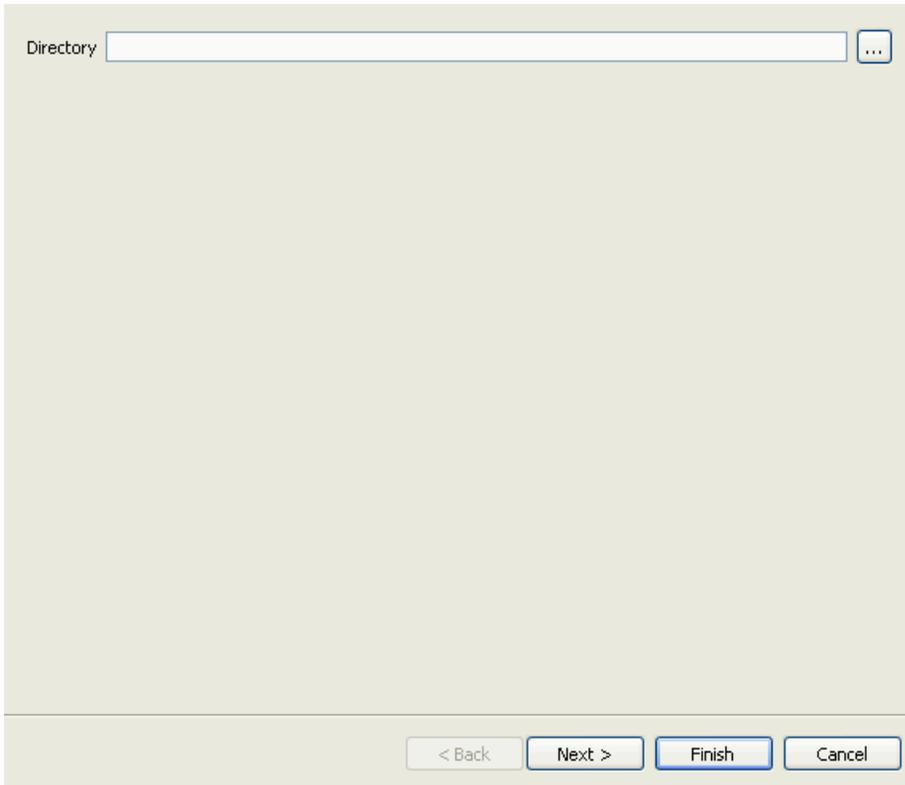
3. Click [...]. The Choose Folder pop-up window displays.

    *   If the `secureapis.xml` file does not exist, and if you want to create this file, browse to the folder where you want to store the the `secureapis.xml` file. Click OK. The system automatically creates the new `secureapis.xml` file in the folder that you selected.

    *   If you want to add the new secure APIs to an existing `secureapis.xml` file, browse to the folder that contains this file. Click OK.

4. Click Next. The Add Secure APIs window displays.

5. In the Add APIs tab:

- If you want to directly add the name of the APIs that needs to be called securely in the `secureapis.xml` file:

  In the API Name, enter the name of the API, or press Ctrl+Space and select the API from the drop-down list and click Add API. The system automatically adds the name of the API to the `secureapis.xml` file.

- If you want to add the name of the commands that needs to be called securely in the `secureapis.xml` file:

  In the API Name, enter the name of the secure API, or press Ctrl+Space and select the API from the drop-down list and click Search.

The Command Name column displays the name of all commands that use the API you specified in the API Name field. The Form ID column displays the identifier of the form in which you are calling this particular command.

> **Note:** To search for commands that use a specific API, specify the API attributes in the Enter the Attributes To Filter Commands Search field. If you are specifying more than one attribute of an API, you must delimit them with a comma.

Click Add Commands. The system automatically adds the name of the commands in the `secureapis.xml` file.

6. Select the Secured APIs tab. You can view the following information:

   - Added APIs—displays the list of APIs that are added to the `secureapis.xml` file.

   - Added Commands—The Command Name column displays the command names that have been added to the `secureapis.xml` file. The Form ID column displays the identifier of the form in which you are calling this particular command.

   - Secure APIs XML—displays the content or structure of the `secureapis.xml` file.

7. Click Finish.

8. Copy the `secureapis.xml` file to the `<RCP_EXTN_FOLDER>/resources` directory.

   where `<RCP_EXTN_FOLDER>` refers to the temporary folder that you created for storing RCP extensions.

   For more information about creating `<RCP_EXTN_FOLDER>` folder and deploying secure APIs in the Sterling Rich Client application, see the *Sterling Supply Chain Applications Installation Guide*.

6.20.1.11 Viewing Shared Tasks

Using the Sterling RCP Extensiblity tool, you can view the shared task details used across applications or plug-ins. You can view the:

- Identifier of the shared task

- Name of the shared task
- A brief description of the shared task
- Template of the input XML model
- Template of the output XML model

To view the shared task details:

1. Start the RCP Extensibility Tool.

   For more information about starting the tool, see Section 6.20.1.1, "Starting the Rich Client Platform Extensibility Tool".

2. Click View Shared Tasks and click once on the screen. The Shared Tasks window displays.

Table 6–44   Shared Tasks Window

Field	Description
Task ID	Displays the unique identifier of each shared task.
Task Name	Displays the name of each shared task.
Description	Displays the description of each shared task.

3. If you want to view more details about a particular shared task, select the shared task and click View Task Details. The Shared Task Details Panel pop-up window displays.

Table 6–45   Shared Task Details Panel Window

Field	Description
Task ID	Displays the unique identifier of the shared task.
Task Name	Displays the name of the shared task.
Description	Displays the description of the shared task.
Input XML	Displays the template structure that is used as an input XML by the shared task.
Output XML	Displays the template structure that is used as an output XML by the shared task.

6.21 Building and Deploying RCP Extensions

After making extensions to a Sterling Rich Client application, make sure that you build and deploy the new extensions. You should build and deploy the Sterling Rich Client application with all the new plug-ins that you created, resource files that you synchronized, and SSL certificates.

This section explains the following:

- Building RCP Extensions

- Deploying RCP Extensions

6.21.1 Building RCP Extensions

To build the RCP extensions:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The plug-in project is displayed in the Navigator view.

3. Right-click on the plug-in project that you want to build and deploy. For more information about creating a plug-in project, see Section 6.4.3, "Creating a Plug-in Project".

4. Select Export… from the pop-up menu. The Export window displays.
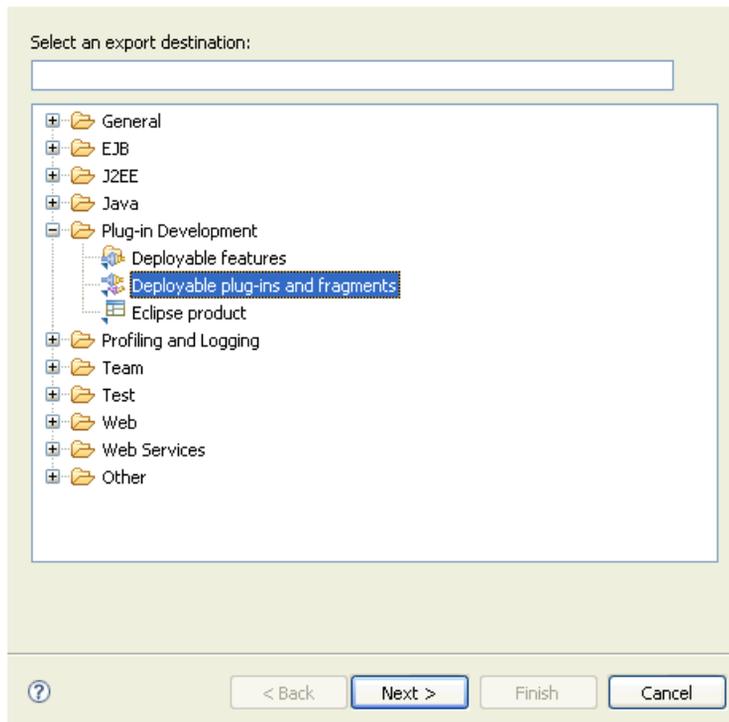
5. From the list of export destinations, under Plug-in Deployment, select Deployable plug-ins and fragments.

6. Click Next.
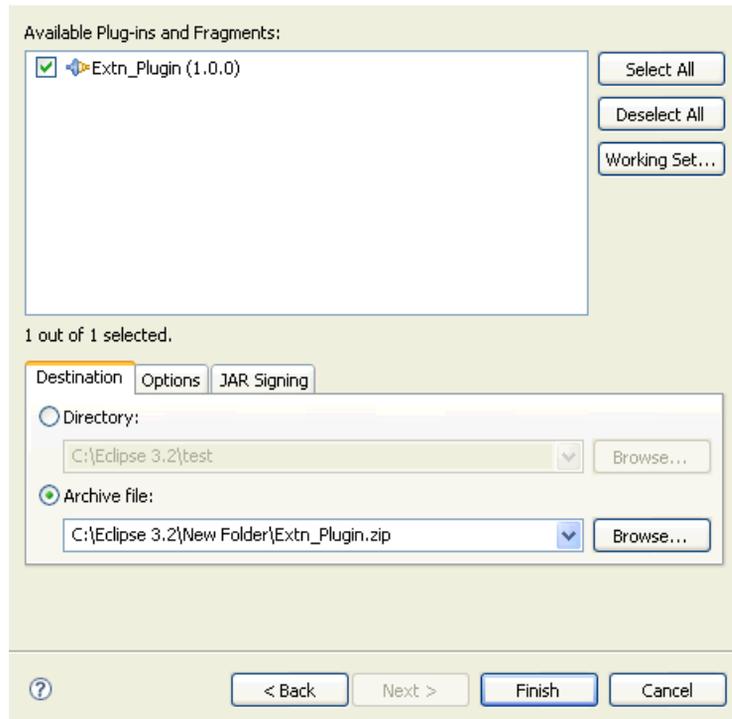
7. In the Destination tab, Choose Archive file: .

8. Click Browse and browse to the folder where you want to store the exported plug-in zip file.

9. In the Options tab, make sure that the Package plug-ins as individual JAR archives box is checked.

10. Click Finish. The plug-in jar is generated and stored in the `plugins` folder in the zip file as specified in Step 8.

6.21.2 Deploying RCP Extensions

After you build the RCP extensions plugin jar, you must deploy this plug-in.

To deploy the RCP extensions:

Copy the plugin jar that you built to the `plugins` directory of the `<RCP_EXTN_FOLDER>` folder and follow the steps as described in the "Deploying And Updating Sterling Rich Client Application" chapter of the *Sterling Supply Chain Applications Installation Guide*.

7

Extending the Sterling Supply Chain Applications Database

Database extensibility enables you to modify tables to maintain more attributes. This chapter explains the Guidelines for Extending Sterling Supply Chain Applications Database as well as how to modify the Sterling Supply Chain Applications database by:

- Adding a Column to a Standard Table
- Adding Unique Tag Identifiers or Descriptors to a Standard Table
- Adding Non-Unique Indices to a Standard Table
- Adding Foreign Key Elements to a Standard Table
- Adding Text Search Indices to a Standard Table
- Creating Custom and Hang-off Tables

Additionally, this chapter explains:

- Generating Audit References for Entities
- Extending API Templates.
- Incorporating Database Extensibility Modifications
- Custom Code Requirements to Avoid Deadlocks
- Merging Documentation Folders

7.1 Guidelines for Extending Sterling Supply Chain Applications Database

Certain aspects of the Sterling Supply Chain Applications database cannot be modified. If you try to make these modifications, your data is not harmed, but your attempted changes are not incorporated into the Sterling Supply Chain Applications database. Sterling Supply Chain Applications do not permit modification of the following:

- Existing columns of tables
- Primary keys of tables
- Unique keys of tables
- Views

When planning extensions to the database, consider the implications of your changes and how they may impact other areas.

> **Important:** If you modify a table and your deployment uses Sterling Supply Chain Analytics, the view associated with the table must also be modified.

Entity Relationship Diagrams

To learn more about the Sterling Supply Chain Applications database, see the entity relationship diagrams (ERDs) using the `<YANTRA_ HOME>/Applications/Foundation/documentation/erd/html/erd.html` file. These ERDs provide the following information:

- Indicate which tables can be extended by adding columns.
- Indicate which tables can have hang-off relationship.
- Relationships between tables (to help you understand the relationship between logical entities such as orders, shipments, and payments).
- Indices details. Each table is indexed by a primary key. Most tables also have a unique index that is constituted of the columns that make the logical unique key. In addition, some tables have alternate indices to support queries.
- Views that indicate how several tables interact.

Entity Database XML Files

The standard tables that are shipped with Sterling Supply Chain Applications are defined in a set of entity XML files, also known as *database definition* XML files. Each entity XML file may contain several table definitions. To learn more about these tables, see the files in the `<YANTRA_HOME>/Applications/Foundation/database/entities/` directory. Within these entity XML files, an entity represents a table and an attribute represents a column.

The following sections describe the general guidelines to follow when adding columns, indices and foreign key elements.

7.1.1 Guidelines for Adding Columns to a Standard Table

When extending the columns of standard tables, keep the following considerations in mind:

- You can only add columns to tables as specified in the ERDs.

- You cannot remove or modify any columns.

- You can add columns either before or after installation of Sterling Supply Chain Applications.

- For all columns added to a Sterling Supply Chain Applications table, you must provide a default value that is relevant to the database framework.

- You cannot use nullable columns for the following fields:

    – Primary Key Attributes

    – Entity Relationships

    Hence, in the entity XML, `Nullable="true"` is allowed for all columns except the ones noted above.

- You cannot add columns with a data type of `Long`.

- When using components of Sterling Supply Chain Applications (such as events and user exits) that read in a map or publish a map (such as the `GetOrderNoUE` user exit), extended fields in the maps are prefixed with `Extn_`.

> **Note:** In Oracle database, the column data type `CLOB` is generated as `LONG` by the Sterling Supply Chain Applications framework.

7.1.2 Guidelines for Adding Non-Unique Indices to a Standard Table

When adding non-unique indices, use a naming convention that differs from Sterling Supply Chain Applications convention, which is `<tablename>_i<1+n>`. Using your own naming convention prevents your indices from accidentally being dropped during upgrades. The following considerations are also recommended:

- Adding a prefix that doesn't start with `Y`.
- Prefix your non-unique indices with `EXTN_` for easier identification.
- Unique indices are not allowed for tables.
- Column names for indices must be valid.
- Index names should not exceed 18 characters.

7.1.3 Guidelines for Adding Foreign Key Elements to a Standard Table

Currently foreign key relationships in extended columns of tables are restricted to only the `YFS_PERSON_INFO` table. When exposing foreign key elements, the following validations are performed:

- The parent table name must be `YFS_PERSON_INFO`.
- The parent column name must be a primary key of the `YFS_PERSON_INFO table`.

7.1.4 Guidelines for Adding Text Search Index Elements to a Standard Table

When adding text search index elements to standard tables, keep the following considerations in mind:

- You can add the text search element only to non-transactional tables.

- The searches performed for the text search indices will be case insensitive.

- You can define more than one text search indices per entity.

- You cannot define multiple columns on a text index.

- Sterling Commerce supports CTXCAT and CONTEXT text search index types on Oracle.

7.2 Extending the Sterling Supply Chain Applications Database Schema

You can extend the Sterling Supply Chain Applications database by:

- Adding a Column to a Standard Table

- Adding Unique Tag Identifiers or Descriptors to a Standard Table

- Adding Non-Unique Indices to a Standard Table

- Adding Foreign Key Elements to a Standard Table

- Adding Text Search Indices to a Standard Table

- Creating Custom and Hang-off Tables

7.2.1 Adding a Column to a Standard Table

You add columns to tables by modifying the entity database extension XML files and then rebuilding the Sterling Supply Chain Applications database and JAR files. After Sterling Supply Chain Applications have been rebuilt, the APIs recognize these added columns and use them when storing and retrieving data.

To add a column to a standard table:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/database/entities/extensions/ Extensions.xml.sample` file to `<your_filename>.xml` or modify your existing extension XML file.

2. Edit the `<your_filename>.xml` file to add a new entity tag as shown in Example 7–1 for each table you want to extend. If the tag already exists, use the existing one. For a description of the XML attributes, see Table 7–1 on page 418.

Example 7–1   Sample XML for Extending Columns

```
<!-- element exposed to create a column -->
<DBSchema>
    <Entities>
        <Entity TableName="REQUIRED">
            <Attributes>
            <Attribute ColumnName="REQUIRED" DataType="" DecimalDigits=""
            DefaultValue="" Description="" Nullable="false" Size="1"
            Type="REQUIRED" XMLName="" XMLGroup="" SqlServerDataType="" />
            </Attributes>
        </Entity>
    </Entities>
</DBSchema>
```

Table 7–1   Attributes in the XML <Attributes> Tag

Attribute	Description
ColumnName	Required. Name of the column added to this table. The ColumnName must start with `EXTN_`.
DataType	Optional. Valid values are available in the `<YANTRA_ HOME>/Applications/Foundation/template/api/Y FSDataTypes.xml` file.
DecimalDigits	Optional. Number of digits of precision required after the decimal. Needed only for numeric fields.
DefaultValue	Required. Used as is for the defaults clause in your database.
Description	Optional. Description of column usage.
Nullable	Optional. Attribute used to describe the nullable value of a field. Default is false. `Nullable=true` is allowed for all columns except Primary Key Attributes and Entity Relationships.
Size	Size of the database column.

Table 7–1   Attributes in the XML <Attributes> Tag

Attribute	Description
Type	Required. Data type of the database column. This attribute also determines the type of attribute in the Java classes that are generated and the format of the attribute in the XML. The valid types are CHAR, VARCHAR2, NUMBER, DATE, and TIMESTAMP.
	If you are using SQL Server and want to specify a data type as TEXT in the database, you also need to use the SqlServerDataType attribute and specify the attribute value as TEXT.
	Note: If DATE is specified, only the calendar date is stored. If TIMESTAMP is specified, the calendar date and time are stored.
XMLName	XML name of the attribute, if it is different from the name of the attribute.
	Choose a name that does not conflict with the base extension. It is recommended that you use Extn as a prefix. It is also strongly recommended that you use the same convention for arriving at the XMLName as the Sterling Supply Chain Applications base product does: Make each letter following the underscore in the column name upper case, and the rest lower case. Then, remove the underscores. Thus, Extn_Item_Id should be: ExtnItemId.
XMLGroup	If present, indicates the child tag in which the attribute is present. If the attribute is not present in the XML, use the NOT_SHOWN string.
	The XMLGroup must be Extn. Thus, the data for the extended columns is in a separate element in the API XML output.
SqlServerDataType	Optional. Pertains only to SQL Server databases. If you see a warning about the row size being too long, specify one or more of your larger columns as "TEXT".
	Columns of type TEXT are not included in the maximum row size calculation for a table.

3. Create a new Attribute tag for each column you want to add to the table.

4. Manually add the columns to the database. You can also use the database verification tool `dbverify` for generating scripts to add columns to your database.

> **Note:** On SQL Server, the total length of all extended columns should not exceed 900 bytes. If SQL Server throws a warning that the row size exceeds the maximum length, change the data type of one or more of your columns to TEXT and then specify TEXT for the SqlServerDataType attribute as described in Table 7–1.

5. Extend the corresponding API templates by following the steps described in Section 7.4, "Extending API Templates".

6. Follow the steps in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

> **Note:** If you are adding columns to extend attributes in the YFS_ITEM table and you want to make these attributes available for classification inheritance, a duplicate entity tag must be added to your XML for the YFS_CLASS_ITEM_ ATTR table.
>
> Additionally, `Nullable` should be set to "`true`" and `DefaultValue` should not be passed for these attributes.
>
> If you do not want to utilize classification inheritance functionality for these attributes, this note can be ignored.
>
> For more information on defining item attributes at the classification level, refer to the *Sterling Product Management Configuration Guide*.

A special case of extending columns for adding unique tag identifiers or descriptors is explained in Section 7.2.2, "Adding Unique Tag Identifiers or Descriptors to a Standard Table".

7.2.2 Adding Unique Tag Identifiers or Descriptors to a Standard Table

The Sterling Supply Chain Applications default tag identifiers are Batch Number, Revision Number, and Lot Number. You may have a need to extend the Sterling Supply Chain Applications Database to define unique tag identifiers or descriptors.

Sterling Commerce recommends that the data type of any unique tag identifiers or descriptors that you add be CHAR or VARCHAR.

> **Note:** Whenever you extend the tag attributes you must also extend the console because the templates for the APIs do not contain these extended tag attributes.

For example, if you work in the metal industry, you may want to use a custom tag identifier named Steel which has both Mill and Grade attributes. Since these are not supplied by default in Sterling Supply Chain Applications, you must extend the set of tables listed below to include the Steel tag identifier column in each table.

7.2.2.1 Extending Tables When Adding Unique Tag Identifiers

You must extend each of the following tables whenever you add unique tag identifiers to the Sterling Supply Chain Applications database:

- YFS_COUNT_RESULT_TAG
- YFS_COUNT_TAG
- YFS_INVENTORY_AUDIT
- YFS_INVENTORY_TAG
- YFS_ITEM_TAG - The data type to be used for this table is CHAR(2).
- YFS_MOVE_REQUEST_LINE_TAG
- YFS_ORDER_KIT_LINE_SCHEDULE
- YFS_ORDER_KIT_LINE_SCHEDULE_H
- YFS_ORDER_LINE_REQ_TAG
- YFS_ORDER_LINE_REQ_TAG_H
- YFS_ORDER_LINE_SCHEDULE

- YFS_ORDER_LINE_SCHEDULE_H

- YFS_ORDER_LINE_RESERVATION

- YFS_RECEIPT_LINE

- YFS_RECEIPT_LINE_H

- YFS_SHIPMENT_LINE_REQ_TAG

- YFS_SHIPMENT_LINE_REQ_TAG_H

- YFS_SHIPMENT_TAG_SERIAL

- YFS_SHIPMENT_TAG_SERIAL_H

- YFS_WORK_ORDER_COMP_TAG

- YFS_WORK_ORDER_COMP_TAG_H

- YFS_WORK_ORDER_TAG

- YFS_WORK_ORDER_TAG_H

7.2.2.2 Extending Tables When Adding Unique Tag Descriptors

You must extend each of the following tables whenever you add unique tag descriptors to the Sterling Supply Chain Applications database:

- YFS_COUNT_RESULT_TAG

- YFS_COUNT_TAG

- YFS_INVENTORY_TAG

- YFS_ITEM_TAG The data type to be used for this table is CHAR(2).

- YFS_ORDER_LINE_REQ_TAG

- YFS_ORDER_LINE_REQ_TAG_H

- YFS_RECEIPT_LINE

- YFS_RECEIPT_LINE_H

- YFS_SHIPMENT_LINE_REQ_TAG

- YFS_SHIPMENT_LINE_REQ_TAG_H

- YFS_SHIPMENT_TAG_SERIAL

- YFS_SHIPMENT_TAG_SERIAL_H

- YFS_WORK_ORDER_COMP_TAG
- YFS_WORK_ORDER_TAG
- YFS_WORK_ORDER_TAG_H

7.2.3 Adding Non-Unique Indices to a Standard Table

You can add non-unique indices to entities. You add indices to a standard Sterling Supply Chain Applications table, by adding an `Index` element in the extension XML for that table.

To add non-unique indices to a standard table:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/database/entities/extensions/ Extensions.xml.sample` file to `<your_filename>.xml` or modify your existing extension XML file.

2. Edit the `<your_filename>.xml` file to add non-unique indices as shown in Example 7–2 for each table you want to extend. For a description of the XML attributes, see Table 7–2 on page 424.

Example 7–2   Sample XML for Adding Non-Unique Indices

```
<!-- element exposed to create index -->
<DBSchema>
<Entities>
   <Entity TableName="REQUIRED">
   .
   .
     <Indices>
        <Index Name="REQUIRED" >
           <Column Name="REQUIRED" />
           .
           .
        </Index>
        .
        .
     </Indices>
     .
     .
   </Entity>
</Entities>
</DBSchema>
```

Table 7–2   Creating Non-Unique Indices on Sterling Supply Chain Applications table

Attribute	Description
Entity	
TableName	Required. Name of the table for which the indices are added, For example: `YFS_ITEM`.
Entity/Index	
Name	Required. The name of the custom index. Name should start with a prefix `EXTN_`
Entity/Index/Column	
Name	Required. The name of the column for which the index is added. Create a new `<Column Name/>` for each column for which the index is added.

3. Create a new `Index` tag for each index you want to add to the column.

4. Extend the corresponding API templates to include the non-unique indices by following the instructions in Section 7.4, "Extending API Templates".

5. Follow the instructions in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

7.2.4 Adding Foreign Key Elements to a Standard Table

A foreign key relationship is a relationship between an extended column in any Sterling Supply Chain Applications table and the `YFS_PERSON_INFO` table. You can create foreign key elements to establish relationship between an extended column and the `YFS_PERSON_INFO` table.

> **Note:** Currently, the `YFS_PERSON_INFO` is the only table which supports a relationship to foreign key extensions within Sterling Supply Chain Applications database.

To add foreign key elements to a standard table:

1. Copy the `<<YANTRA_ HOME>/Applications/Foundation/database/entities/extensions/`

Extensions.xml.sample file to <your_filename>.xml or modify your existing extension XML file.

2. Edit the <your_filename>.xml file to add foreign key elements as shown in Example 7–3 for each table you want to extend. For a description of the XML attributes, see Table 7–3 on page 425.

Example 7–3   Sample XML for Adding Foreign Key Elements

```
<!-- element exposed to create foreign key relationship -->
<DBSchema>
  <Entities>
       <Entity TableName="REQUIRED">
        .
        .
        <!-- element exposed to create relationship with PERSON_INFO table -->
          <ForeignKeys>
            <ForeignKey ParentTableName="YFS_PERSON_INFO"
            XMLName="YFSName1" >
              <Attribute ColumnName="REQUIRED"
              ParentColumnName="PERSON_INFO_KEY" />
            </ForeignKey>
            <ForeignKey ParentTableName="YFS_PERSON_INFO"
            XMLName="YFSName2" >
              <Attribute ColumnName="REQUIRED"
              ParentColumnName="PERSON_INFO_KEY" />
            </ForeignKey>
        .
        .
          </ForeignKeys>
        .
        .
       </Entity>
     </Entities>
   </DBSchema>
```

Table 7–3   Creating ForeignKey elements on YFS_PERSON_INFO table

Attribute	Description
Entity	
TableName	Required. Name of the table for which the foreign key elements are added; For example: YFS_ITEM.
Entity/ForeignKeys/ForeignKey	

Table 7–3   Creating ForeignKey elements on YFS_PERSON_INFO table

Attribute	Description
ParentTableName	The name of the parent table for this foreign key element. **Note**: This value must be YFS_PERSON_INFO which is the only table that currently supports foreign key relationships.
XMLName	You can specify the XML representation of the element name. It must start with the prefix of the parent entity. For example, if ParentTableName is prefixed with YFS then the XMlName must start with YFS. By default the parent table name is assumed.
Entity/ForeignKeys/ForeignKey/Attribute	
ColumnName	Specifies the extended column name of the Entity.
ParentColumnName	The column name of the YFS_PERSON_INFO that has a foreign key element relationship.

3. Create a new ForeignKey tag for each foreign key relationship you want to add.

4. Multiple foreign key elements can be related to the same parent table.

5. Extend the corresponding API templates to include the foreign key elements by following the instructions in Section 7.4, "Extending API Templates".

6. Follow the instructions in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

7.2.5 Adding Text Search Indices to a Standard Table

You can add text search indices to entities. You add text search indices to a standard Sterling Supply Chain Applications table, by adding an TSIndex element in the extension XML for that table.

To add text search indices to a standard table:

1. Copy the <YANTRA_ HOME>/Applications/Foundation/database/entities/extensions/

Extensions.xml.sample file to <your_filename>.xml or modify your existing extension XML file.

2. Edit the <your_filename>.xml file to add text search indices as shown in Example 7–4 for each table you want to extend. For a description of the XML attributes, see Table 7–4 on page 424.

Example 7–4 Sample XML for Adding Text Search Indices

```
<!-- element exposed to create index -->
<DBSchema>
<Entities>
   <Entity TableName="REQUIRED">
   .
   .
      <TSIndices>
         <TSIndex Name="REQUIRED" >
            <Column Name="USERNAME" />
         </TSIndex>
         .
         .
      </TSIndices>
      .
      .
   </Entity>
</Entities>
</DBSchema>
```

Table 7–4 Creating Text Search Indices on Sterling Supply Chain Applications table

Attribute	Description
Entity	
TableName	Required. Name of the table for which the text search indices are added. For example: YFS_USER.
Entity/TSIndex	
Name	Required. The name of the text search index. For example: YFS_TS_USER_Name.
Entity/TSIndex/Column	
Name	Required. The name of the column for which the text search index is added. You cannot define multiple columns on a text index.

3. Create a new TSIndex tag for each text search index you want to add to the column.

4. Follow the instructions in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

7.2.6 Creating Custom and Hang-off Tables

With the Sterling Supply Chain Applications Database Framework you can also extend the Sterling Supply Chain Applications database by creating custom or hang-off tables.

A **custom table** is an independent table and cannot be modeled as an extension to standard Sterling Supply Chain Applications table.

A **hang-off** table is a table with many to one relationship with a standard Sterling Supply Chain Applications table.

Creating a custom or hang-off entity enables you to:

- Create a relationship between a Sterling Supply Chain Applications standard table and a hang-off table.

- Invoke Extensible APIs that store and retrieve data from hang-off tables.

- Invoke dbverify for generating appropriate SQL scripts to create or alter tables for custom or hang-off entities.

- Audit item and organization tables.

The ability to purge data from hang-off tables is discussed in Section 7.2.6.2.1. Keep in mind the following which apply to the creation of custom or hang-off tables:

- You can only determine if an entity is enabled for hang-off by referencing the associated Entity Relationship Diagram (ERD) located in the <YANTRA_ HOME>/Applications/Foundation/documentation/ERD directory.

- Currently only order, order line, work order, shipment, item, and organization tables are marked as hang-off enabled.

- Custom and hang-off table names must not start with a Y.

- The "Extn" part is trimmed off from the XML name of the custom and hang-off tables.

- Primary key name must not start with a `Y`.

- Entity names must start with the prefix provided in the entity definition.

- The YIFApi interface does not extend APIs for custom/hang-off tables. Therefore, the APIs for these tables must be configured as services.

- Javadocs are not created for the APIs created by the infrastructure to support custom and hang-off tables.

  However, you can create javadocs or ERDs for your custom or hang-off tables by running a documentation merging tool discussed in Section 7.7, "Merging Documentation Folders".

- XSD generation and validation is not done for custom or hang-off tables.

- Every custom or hang-off entity must have a primary key.

- Every custom or hang-off entity must have the following columns described in Table 7–5:

Table 7–5   Required Columns for Custom or Hang-off Tables

Column Name	Data Type	Default Value
Key-Column	Key	' ' (space)
CREATETS	TimeStamp	sysdate
MODIFYTS	TimeStamp	sysdate
CREATEUSERID	UserId	' ' (space)
MODIFYUSERID	UserId	' ' (space)
CREATEPROGID	ProgramID	' ' (space)
MODIFYPROGID	ProgramID	' ' (space)
LOCKID	Lockid	0 (zero)

7.2.6.1 Steps to create a custom table

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/database/entities/extensions/ Extensions.xml.sample` file to `<your_filename>.xml` or modify your existing extension XML file. For example, assume that `ABC_ CUSTOMER_ORDER_LINE` is a custom table.

2. Edit the `<your_filename>.xml` file to create custom tables as shown in Example 7–5. For a description of the XML attributes, see Table 7–6 on page 432.

Example 7–5   Sample XML for Creating Custom Tables

```
<DBSchema>
    <Entities>
        <Entity ApiNeeded="Y/N" AuditRequired="Y" Description=""
        HasHistory="True/False" Prefix="ABC"
        TableName="ABC_CUSTOMER_ORDER_LINE" >
        <!-- table columns -->
        <Attributes>
            <Attribute ColumnName="CREATETS" DataType="TimeStamp"
            DefaultValue="sysdate" Description="Create TimeStamp" />
            <Attribute ColumnName="MODIFYTS" DataType="TimeStamp"
            DefaultValue="sysdate" Description="Modify TimeStamp" />
            <Attribute ColumnName="CREATEUSERID" DataType="UserId"
            DefaultValue="&apos; &apos;" Description="Creating User ID" />
            <Attribute ColumnName="MODIFYUSERID" DataType="UserId"
            DefaultValue="&apos; &apos;" Description="Modifying User ID" />
            <Attribute ColumnName="CREATEPROGID" DataType="ProgramID"
            DefaultValue="&apos; &apos;" Description="Creating Program ID" />
            <Attribute ColumnName="MODIFYPROGID" DataType="ProgramID"
            DefaultValue="&apos; &apos;" Description="Modifying Program ID" />
            <Attribute ColumnName="LOCKID" DataType="Lockid" DefaultValue="0"
            Description="Lock ID" />
            <Attribute ColumnName="TABLE_KEY" DataType="Key" DefaultValue=" "
            Description="" Nullable="True/False" XMLName="TableKey" />
            .
            .
        </Attributes>
        <!-- PrimaryKey is a mandatory attribute in entity definition. This
        element can have ONLY ONE attribute element -->
        <PrimaryKey Name="TABLE_NAME_PK">
            <Attribute ColumnName="TABLE_KEY" />
        </PrimaryKey>
        <!-- Indices -->
        <Indices>
            <Index Name="INDEX_I1" Unique="True/False" >
            <Column Name="Attribute2" />
            .
            .
            </Index>
            .
```

```
                    .
        </Indices>
        <!-- Relationship -->
        <Parent ParentTableName="YFS_ORDER_LINE" XMLName="YFSOrderLine" >
            <Attribute ColumnName="CUSTOM_ORDER_KEY" ParentColumnName="ORDER_
            LINE_KEY" />
            .
            .
        </Parent>
        <!-- ForeignKeys -->
        <ForeignKeys>
            <ForeignKey ParentTableName="PARENT_ORDER_LINE"
            XMLName="PARENTName1" >
            <Attribute ColumnName="CUSTOM_ORDER_KEY" ParentColumnName="PARENT_
            COLUMN_KEY" />
            .
            .
            </ForeignKey>
            .
            .
        </ForeignKeys>
        <!-- AuditReferences -->
        <AuditReferences>
            <Reference ColumnName="TABLE_KEY" />
            .
            .
        </AuditReferences>
        </Entity>
    </Entities>
</DBSchema>
```

3. The following table explain the attributes in the entity XML:

Table 7–6   Entity XML Definitions for Custom Tables

Attribute	Description
Entity	
ApiNeeded	Indicate whether or not APIs should be generated. Valid values are `Y` or `N`. A default set of API's are generated if `Y` is passed.
	For example in the `ABC_CUSTOMER_ORDER_LINE` table, Sterling Supply Chain Applications create following API's when the database extension jar file is generated:
	`listABCCustomerOrderLine()`
	`getABCCustomerOrderLine()`
	`createABCCustomerOrderLine()`
	`modifyABCCustomerOrderLine()`
	`deleteABCCustomerOrderLine()`
	These APIs can be accessed as services using Sterling Service Definition Framework. For more information see, Section 7.4.3, "Configuring Services for Custom and Hang-off APIs".
AuditRequired	If set to `Y` audit record for this entity are created. Generating audit for entities is described in Section 7.3, "Generating Audit References for Entities".
HasHistory	This flag indicates whether the custom table can have an history table associated with it.
	The default value is "False".
	If the flag is set to "True", the appropriate scripts for generating database scripts for creating and altering the history table is generated by dbverify.
	For a custom table, the HasHistory flag must be set to "True" for generating history tables. However, if a Parent relationship is defined in the entity XML, this flag is copied from the parent table definition, and all child entities cannot override this flag.
Prefix	The prefix added to your custom tables. It is recommended that you do not use a prefix starting with "Y".
TableName	The name given to your custom table.

Table 7–6   Entity XML Definitions for Custom Tables

Attribute	Description
Entity/Attributes/Attribute	
ColumnName	The names of the column that comprise the table.
DataType	The data type of the column. Valid data types are given in `<YANTRA_ HOME>/Applications/Foundation/template/api/YFSDataTypes.xml` file.
DefaultValue	Default value for the column.
Description	A description of the columns that could be used in javadocs or ERD.
Nullable	Optional. Attribute used to describe the nullable value of a field. Default is false. `Nullable=true` is allowed for all columns except Primary Key Attributes and Entity Relationships.
XMLName	Optional. XML name of the attribute, if it is different from the name of the attribute.
	Choose a name that does not conflict with the base extension. It is recommended that you use `Extn` as a prefix. It is also strongly recommended that you use the same convention for arriving at the XMLName as the Sterling Supply Chain Applications base product does: Make each letter following the underscore in the column name upper case, and the rest lower case. Then, remove the underscores. Thus, `Extn_Item_Id` should be: `ExtnItemId`.
Entity/PrimaryKey	
Name	Name of the unique index created for the primary key. This value cannot exceed 18 characters.
	Note: The name of the primary key in the extenstion XML should end with `_PK`.
ColumnName	The name of the table column that is identified as the primary key.
Entity/Indices/Index	
Name	The index name. This value cannot exceed 18 characters.

Table 7–6   Entity XML Definitions for Custom Tables

Attribute	Description
Unique	This key is present only for custom entities. Valid values are `True` or `False`. If `True` an unique index is created.
Column/ Name	The table column name associated with the index.
Entity/Parent	
ParentTableName	Name of the other table this entity has foreign key relationship.
XMLName	The XML name of the parent attribute. It should start with the prefix mentioned in the parent table. By default the parent table name is assumed.
Parent/Attribute Level	
ParentColumnName	Column name in the parent table. Note: To create relationships among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this custom entity.
Entity/ForeignKeys/ForeignKey	
ParentTableName	The name of the table with which the entity has a foreign key relationship.
XMLName	XML representation of the element name. By default the parent table name is assumed.
Entity/ForeignKeys/ForeignKey/Attribute	
ParentColumnName	Column name of the parent table. Note: To create foreign keys among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this custom entity.
Entity/AuditReferences/Reference	
ColumnName	Reference Column name in the audit table.

> **Note:** In entity definition, relationship can be defined under `Parent` and `ForeignKey` elements.

4. The relationship defined under the `ForeignKey` element indicates:

    a. If the foreign table is a Sterling Supply Chain Applications table, for a single record in the foreign table, zero or many records in this custom table may exist.

    b. This is a read-only relationship, hence deletion of a record from the foreign table does not result in the deletion of a matching record from this custom table.

5. The relationship defined under the `Parent` element indicates:

    a. For a single record in the parent table, multiple child records may exist.

    b. Deletion of a record from the parent table will result in the deletion of matching records from the child table, if any.

6. Extend the corresponding API templates (for example, `getOrderDetails()`API) by following the instructions in Section 7.4, "Extending API Templates".

> **Note:** The APIs generated by the Sterling Supply Chain Applications for the custom tables can be invoked only as a service. For more information see Section 7.4.3, "Configuring Services for Custom and Hang-off APIs" on page 448.

7. Follow the instructions in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

7.2.6.2 Steps to create a hang-off table

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/database/entities/extensions/ Extensions.xml.sample` file to `<your_filename>.xml` or modify your existing extension XML file. For example, assume that `ABC_ CUSTOMER_ORDER_LINE` is an hang-off table.

2. Edit the <your_filename>.xml file to create hang-off tables as shown in Example 7–6. For a description of the XML attributes, see Table 7–7 on page 432.

Example 7–6   Sample XML for Creating Hang-off Tables

```
<DBSchema>
   <Entities>
      <Entity ApiNeeded="Y/N" AuditRequired="Y" Description=""
        HasHistory="True/False" Prefix="ABC"
        TableName="ABC_CUSTOMER_ORDER_LINE" >
        <!-- table columns -->
        <Attributes>
            <Attribute ColumnName="CREATETS" DataType="TimeStamp"
            DefaultValue="sysdate" Description="Create TimeStamp" />
            <Attribute ColumnName="MODIFYTS" DataType="TimeStamp"
            DefaultValue="sysdate" Description="Modify TimeStamp" />
            <Attribute ColumnName="CREATEUSERID" DataType="UserId"
            DefaultValue="&apos; &apos;" Description="Creating User ID" />
            <Attribute ColumnName="MODIFYUSERID" DataType="UserId"
            DefaultValue="&apos; &apos;" Description="Modifying User ID" />
            <Attribute ColumnName="CREATEPROGID" DataType="ProgramID"
            DefaultValue="&apos; &apos;" Description="Creating Program ID" />
            <Attribute ColumnName="MODIFYPROGID" DataType="ProgramID"
            DefaultValue="&apos; &apos;" Description="Modifying Program ID" />
            <Attribute ColumnName="LOCKID" DataType="Lockid" DefaultValue="0"
            Description="Lock ID" />
            <Attribute ColumnName="TABLE_KEY" DataType="Key" DefaultValue=" "
            Description="" Nullable="True/False" XMLName="TableKey" />
            .
            .
        </Attributes>
        <!-- PrimaryKey is a mandatory attribute in entity definition. This
        element can have ONLY ONE attribute element -->
         <PrimaryKey Name="TABLE_NAME_PK">
            <Attribute ColumnName="TABLE_KEY" />
         </PrimaryKey>
        <!-- Indices -->
         <Indices>
            <Index Name="INDEX_I1" Unique="True/False" >
            <Column Name="Attribute2" />
            .
            .
            </Index>
            .
```

```
               .
           </Indices>
        <!-- Relationship -->
         <Parent ParentTableName="YFS_ORDER_LINE" XMLName="YFSOrderLine" >
            <Attribute ColumnName="CUSTOM_ORDER_KEY"
            ParentColumnName="ORDER_LINE_KEY" />
            .
            .
         </Parent>
         <ForeignKeys>
            <ForeignKey ParentTableName="PARENT_ORDER_LINE"
            XMLName="PARENTName1" >
               <Attribute ColumnName="CUSTOM_ORDER_KEY"
               ParentColumnName="PARENT_COLUMN_KEY" />
             .
             .
            </ForeignKey>
             .
             .
            </ForeignKeys>
        <!-- AuditReferences -->
         <AuditReferences>
            <Reference ColumnName="TABLE_KEY" />
            .
            .
         </AuditReferences>
      </Entity>
   </Entities>
</DBSchema>
```

3. The following table explain the attributes in the entity XML:

Table 7–7   Entity XML Definitions for Hang-off Tables

Attribute	Description
Entity	
ApiNeeded	Indicate whether or not APIs should be generated. Valid values are `Y` or `N`. A default set of API's are generated if `Y` is passed.
	For example in the `ABC_CUSTOMER_ORDER_LINE` table, Sterling Supply Chain Applications create following API's when the database extension jar file is generated:
	`listABCCustomerOrderLine()`
	`getABCCustomerOrderLine()`
	`createABCCustomerOrderLine()`
	`modifyABCCustomerOrderLine()`
	`deleteABCCustomerOrderLine()`
	These APIs can be accessed as services using Sterling Service Definition Framework. For more information see, Section 7.4.3, "Configuring Services for Custom and Hang-off APIs".
AuditRequired	If set to `Y` audit record for this entity are created. Generating audit for entities is described in Section 7.3, "Generating Audit References for Entities".
	Note: This attribute must not be passed when you are creating a hang-off for order related tables. In this case, the audits are automatically inserted into the `YFS_ORDER_AUDIT` table.
Description	A description of the entity that could be used in javadocs or ERD.
HasHistory	This flag is automatically inherited from the parent table. For example, let us assume that `ABC_ORDER_HEADER` table is created as an hang-off table for `YFS_ORDER_HEADER`, which has an associated history table. Then `ABC_ORDER_HEADER_H` is automatically generated by the database framework.
Prefix	The prefix added to your custom tables. It is recommended that you do not use a prefix starting with `Y`.

Table 7–7   Entity XML Definitions for Hang-off Tables

Attribute	Description
TableName	The name given to your hang-off table.
Entity/Attributes/Attribute	
ColumnName	The names of the column that comprise the table.
DataType	The data type of the column. Valid data types are given in `<YANTRA_HOME>/Applications/Foundation/template/api/YFSDataTypes.xml` file.
DefaultValue	Default value for the column
Description	A description of the columns that could be used in javadocs or ERD.
Nullable	Optional. Attribute used to describe the nullable value of a field. Default is false. `Nullable=true` is allowed for all columns except Primary Key Attributes and Entity Relationships.
XMLName	Optional. XML name of the attribute, if it is different from the name of the attribute.
	Choose a name that does not conflict with the base extension. It is recommended that you use `Extn` as a prefix. It is also strongly recommended that you use the same convention for arriving at the XMLName as the Sterling Supply Chain Applications base product does: Make each letter following the underscore in the column name upper case, and the rest lower case. Then, remove the underscores. Thus, `Extn_Item_Id` should be: `ExtnItemId`.
Entity/PrimaryKey	
Name	Name of the unique index created for the primary key. This value cannot exceed 18 characters.
	Note: The name of the primary key in the extension XML should end with `_PK`.
ColumnName	The name of the table column that is identified as the primary key.
Entity/Indices/Index	
Name	The index name. This value cannot exceed 18 characters.

Table 7–7   Entity XML Definitions for Hang-off Tables

Attribute	Description
Unique	This key is present only for custom entities. Valid values are `True` or `False`. If `True` an unique index is created.
Column/ Name	The table column name associated with the index.
Entity/Parent	
ParentTableName	Name of the other table this entity has foreign key relationship.
XMLName	The XML name of the parent attribute. It should start with the prefix mentioned in the parent table.
	By default the parent table name is assumed.
Parent/Attribute Level	
ParentColumnName	Column name in the parent table.
	Note: To create relationships among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this custom entity.
Entity/ForeignKeys/ForeignKey	
ParentTableName	The name of the table with which the entity has a foreign key relationship.
XMLName	XML representation of the element name.
	By default the parent table name is assumed.
Entity/ForeignKeys/ForeignKey/Attribute	
ParentColumnName	Column name of the parent table.
	Note: To create foreign keys among entities, the data type of parent column must be of type `CHAR` or `VARCHAR`.
ColumnName	Column name in this hang-off entity.
Entity/AuditReferences/Reference	
ColumnName	Reference Column name in the audit table.

> **Note:** In entity definition, relationship can be defined under `ForeignKey` elements.

4. The relationship defined under the `ForeignKey` element indicates:

   a. If the foreign table is a Sterling Supply Chain Applications table, for a single record in the foreign table, zero or many records in this hang-off table may exist.

   b. This is a read-only relationship, hence deletion of a record from the foreign table does not result in the deletion of a matching record from this hang-off table.

5. Extend the corresponding API templates (for example, `getOrderDetails` API) by following the instructions in Section 7.4, "Extending API Templates".

   > **Note:** The APIs generated by the Sterling Supply Chain Applications for the hang-off tables can be invoked only as a service. For more information see Section 7.4.3, "Configuring Services for Custom and Hang-off APIs" on page 448.

6. Follow the instructions in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

7.2.6.2.1 Purging Data from Hang-Off Tables

Currently, the Purge agent moves records to history tables. With the custom or hang-off entities enabled, the Purge agent also deletes records from hang-off tables. However, the data from a hang-off table can be purged only if its parent elements are also purged. If a history table exists, records are added to the history table. The records are deleted from the history table using the History Purge agent.

In order to purge the custom and hang-off entities you need to include the `yfsdbextn.jar` file in the classpath of the agent server. For more information on setting up an agent server, see *Sterling Supply Chain Applications Installation Guide*.

7.3 Generating Audit References for Entities

If the `AuditRequired` flag is enabled in the entity XML, audit records are added to the `YFS_AUDIT` table. The default for this flag is `Y`, for item and organization tables. However, the audit flag and audit references can be overridden by the extension XML file.

Only item and organization header-level audit records are inserted in the `YFS_AUDIT_HEADER` table. The audit references refer to the columns of the entity being audited.

The audits can be generated for the hang-off and custom tables, by modifying the entity table name and audit reference column names.

To generate audit references for entities:

1. Edit the `<your_filename>.xml` file in the `<YANTRA_HOME>/Applications/Foundation/database/entities/extensions` directory to enable audit record generation for desired entities. The following example explains the elements to be added to the database schema:

Example 7–7   Sample XML for Creating Audit References

```
<DBSchema>
   <Entities>
      <Entity TableName="YFS_ITEM" AuditRequired="Y" >
         .
         .
         <AuditReferences>
            <Reference ColumnName="ItemId" />
            .
            .
         </AuditReferences>
         .
         .
      </Entity>
   </Entities>
</DBSchema>
```

Table 7–8   Generating audits for entities

Attribute	Description
Entity	

Table 7–8   Generating audits for entities

TableName	The table name to be audited.
AuditRequired	If this flag is set to Y the audit references are entered in the `YFS_AUDIT` table.
	Note: This attribute must not be passed when you are creating a hang-off for order related tables. In this case, the audits are automatically inserted into the related order audit tables.
Entity/AuditReferences/Reference	
ColumnName	The column name in this entity which has audit references. This name must be valid for the entity.

2. Create a new `Reference` tag for each audit reference you want to add.

3. The hang-off of an order table audits can be viewed with the associated order audits.

For example, the audit entries for `AAOrderLine` which is an hang-off of `YFS_ORDER_HEADER` table can be viewed with the Order Audit Details screen as shown below:

For more information on the field details of the order audits, see *Sterling Distributed Order Management User Guide*.

4. The audits for custom tables and hang-off tables not related to the Sterling Supply Chain Applications order entities are stored in `YFS_AUDIT` table and can be obtained using `getAuditList` API.

5. Follow the instructions in Section 7.5, "Incorporating Database Extensibility Modifications" to implement your modifications.

7.4 Extending API Templates

Each template-based API delivers different output, depending on the template passed to it. To verify whether an API is template-based or not, see the *Sterling Supply Chain Applications Javadocs*.

If your table modifications impact any APIs, you must extend the templates of those APIs. Place the extended API templates in the `<YANTRA_HOME>/Applications/Foundation/template/api/extn` directory. For information on extending API templates by creating custom templates, see Section 8.2.6, "API Output XML Files" on page 468.

To find out which APIs are impacted by table modifications:

1. Note the `XMLName` attribute of the table being modified in the `entity` tag inside the database entity XML files (which contains the definition of all the tables). These database entity XML files are located in `<YANTRA_HOME>/Applications/Foundation/Database/entities` directory.

2. Search for the pattern of that `XMLName` attribute in the `<YANTRA_HOME>/Applications/Foundation/template/api` directory. The search, results in finding exposed and internal APIs impacted by the table modifications or extensions.

For example, consider that you want to extend an attribute in the `YFS_CHARGE_CATEGORY` table. The `XMLName` for this table as specified in `<YANTRA_HOME>/Applications/Foundation/Database/entities/omp_tables.xml` is `ChargeCategory`. Now search for the attribute `ChargeCategory` in `<YANTRA_HOME>/Applications/Foundation/template/api` directory to find the APIs impacted by this extension.

7.4.1 Including Extended Attributes in the API Template

The extended attributes appear as a separate `<Extn>` element under the primary element.

For example, in the default output XML template of the `getItemDetails()` API, the `Item` attributes have the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Item .. Item attributes >
    <PrimaryInformation .... PrimaryInformation attributes />
    <ItemServiceSkillList .. ItemServiceSkillList attributes/>
```

```
<ItemAliasList ... ItemAliasList attributes />
.
.
.
</Item>
```

After extending the `Item` header, the `getItemDetails()` API can output the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Item .. Item attributes >
    <PrimaryInformation .... PrimaryInformation attributes />
    <Extn ExtnAltQty="200408201034469490" ..... extnded attributes />
    <YFSPersonInfo .... PersonInfoKey="200408201034469490" ..... />
    <ItemServiceSkillList .. ItemServiceSkillList attributes/>
    <ItemAliasList ... ItemAliasList attributes />
.
.
.
</Item>
```

> **Note:** Foreign Key variables for the extended column
> appear as a `PersonInfoKey` attribute of the
> `YFSPersonInfo` element. The relationship can be validated
> if the extended column and the `PersonInfoKey` have the
> same value.

The extended attribute is retrieved from the `XMLName` attribute of the `<your_filename>.xml` file that you edited in the previous sections, when extending a standard table. Place your extended templates in the `<YANTRA_HOME>/Applications/Foundation/template/api/extn` directory.

7.4.2 Including Custom and Hang-Off Entities in the API Template

The standard APIs can be extended to provide information from the custom or hang-off tables. A tool specifically provided for generating the template XML's, `templateXmlGen.xml` is located in the `<YANTRA_HOME>/Applications/Foundation/bin` directory.

1. Run the template XML generation tool from your `<YANTRA_HOME>/Applications/Foundation` directory by using the following command:

```
ant -Dtable=<TABLE_NAME> -f bin/templateXmlGen.xml
```

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

2. Once the command is executed, the sample XML files are placed in the `<YANTRA_HOME>/Applications/Foundation/extn/sampleXML` directory as `<TABLE_NAME>_sample.xml`.

For example, consider `HF_Order_Header` is an hang-off of `YFS_ Order_Header` table. The generated `HF_Order_Header_sample.xml` is as follows:

```
<HFOrderHeader Createprogid=" " Description=" " DocumentType=" "
EnterpriseKey=" "OrderHeaderKey=" " OrderName=" " OrderNo=" " .... >
</HFOrderHeader>
```

3. A sample XML for including the above attributes in a standard API can be generated by passing the `YFS` table that has relationship with the hang-off table you are interested.

For example, assume `HF_Order_Header` is a hang-off table with a relationship to the `YFS_Order_Header` table. The XML template generated by the tool when `TABLE_NAME=YFS_Order_Header` is passed:

```
<Order>
   <OrderLines>
      <OrderLine ........>
         <Extn extended attributes >
            <HFOrderHeaderList>
               <HFOrderHeader Createprogid=" " Description=" " ......... >
               </HFOrderHeader>
            </HFOrderHeaderList>
         </Extn>
      </OrderLine>
   </OrderLines>
</Order>
```

> **Note:** You can modify the attributes only within your custom or hang-off element.

You can prune this sample XML to include your custom attributes in an API template, such as `getOrderDetails` output template. However, you cannot modify any of the `YFS` elements or attributes.

> **Note:** The sample XMLs are also automatically generated when you create the database extension jar file, and are posted in the `<YANTRA_ HOME>/Applications/Foundation/extn/sampleXML` directory. However, if you need to create a sample template as described in Step 3, you must run the template XML generation tool separately by specifying the corresponding `YFS` table name.

4. An hang-off table can be deleted by passing an Operation attribute in the `change` or `modify` APIs. For example, `HF_Order_Header` element can be deleted in a `changeOrder` API as:

```
<Order>
   <OrderLines>
      <OrderLine ........>
         <Extn extended attributes >
            <HFOrderHeaderList>
               <HFOrderHeader Operation="Delete" Createprogid=" " ....... >
               </HFOrderHeader>
            </HFOrderHeaderList>
         </Extn>
      </OrderLine>
   </OrderLines>
</Order>
```

The operations such as **Create** and **Modify** are executed by default. If an entry for that element exists, the API modifies the entries with the recent value. In the case where that element does not exist it creates a new entry.

5. The records in a hang-off table can be reset by assigning the value of "true" to the Reset attribute at the list level element of the XML. When the records are reset, all existing records for that hang off table that correspond to the parent table are deleted and all elements included under the list element are inserted. For example, hang off records in the HF_Order_Header_list element can be reset by using the following:

```
<Order>
   <OrderLines>
      <OrderLine ........>
         <Extn extended attributes >
```

```
                          <HFOrderHeaderList Reset="true">
                              <HFOrderHeader>
                              </HFOrderHeader>
                          </HFOrderHeaderList>
                      </Extn>
                  </OrderLine>
              </OrderLines>
          </Order>
```

   6. Follow the instructions in Section 7.5, "Incorporating Database
   Extensibility Modifications" to implement your modifications.

7.4.3 Configuring Services for Custom and Hang-off APIs

   The APIs generated for custom or hang-off entities by the Sterling Supply
   Chain Applications can be invoked only as a service. The service
   configuration user interface has to be enabled to configure these APIs.
   For more information on creating a service definition see the *Sterling
   Supply Chain Applications Platform Configuration Guide*.
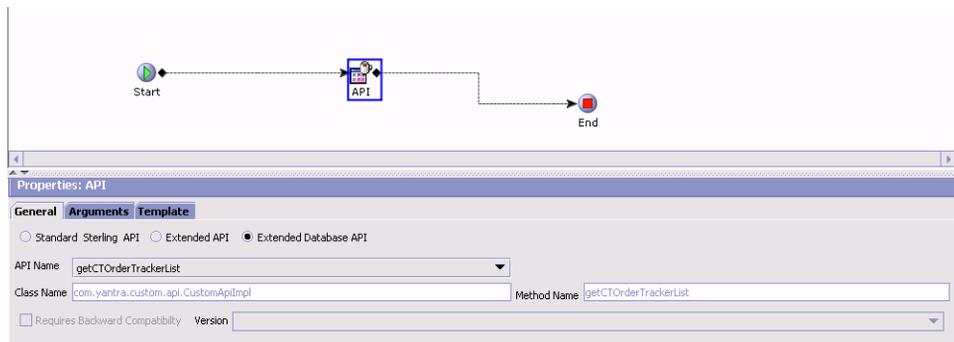
   For including custom APIs you can create a service definition as shown in
   the figure. The configuration fields are explained in Table 7–9.

Table 7–9   API Configuration Properties

Field Name	Description
General Tab	
Standard Sterling Supply Chain Applications API	Select this option if a standard Sterling Supply Chain Applications API is to be invoked. If selected, a Standard Sterling Supply Chain Applications API Name drop down list is displayed. For each API, the Class Name and Method Name are provided and cannot be edited.
Extended API	Select this option if a custom java code is to be invoked.
Extended Database API	Select this option if the service invokes a custom or hang-off API. If selected, a custom API Name drop-down list is displayed. For each API, the Class Name and Method Name are provided and cannot be edited.
API Name	Select the custom API to be invoked.
Class Name	Specifies the class to be called. This field cannot be edited.
Method Name	Specifies the method to be called. This field cannot be edited.
Requires Backward Compatibility	This field is not supported in this release.
Version	This drop-down list specifies the version supported for backward compatibility. This cannot be edited as it is not supported in this release.
Arguments Tab	
Argument Name	You can pass name/value pairs to the API by entering the values in the Arguments Tab.
	In order for custom APIs to access custom values, the API should implement the interface `com.yantra.interop.japi.YIFCustomApi`.
	If entered, these name/value pairs are passed to the `YIFCustomApi` interface as a properties object.
Argument Value	Enter the argument value to be passed.
Template Tab	

Table 7−9   API Configuration Properties

Field Name	Description
XML Template	When the custom APIs are invoked, you can specify an output template to be used by the API.
	Select this option to construct a new XML file to be used for the API output. Enter the template root element name and click OK. You can then construct the XML.
File Name	Select this option to enter the filename of an existing XML file to be used as the API output template. This file should also exist in your CLASSPATH.

7.5 Incorporating Database Extensibility Modifications

After making all of your required database extensibility changes, you need to use the ANT utility to generate a `yfsdbextn.jar` file and then use that JAR file to generate a `yantra.ear` file.

To incorporate database extensibility modifications:

1. Place extension files into the `<YANTRA_HOME>/Applications/Foundation/Database/Entities/Extensions` directory, if they are not already there.

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

2. Recreate the Sterling Supply Chain Applications Runtime. For more information about creating Sterling Supply Chain Applications Runtime, refer to *Sterling Supply Chain Applications Installation Guide*.

3. Execute the `dbverify` tool from `<YFS_HOME>/bin/dbverify.sh` (`.cmd` in Windows) to generate the SQL scripts required for your custom and hang-off tables.

    where <YFS_HOME> refers to the `<YANTRA_HOME>/Runtime` directory.

> **Note:** For more information on using `dbverify` see the
> *Sterling Supply Chain Applications Installation Guide*.

4. Set or export the `ANT_OPTS` environment variable based on your
   operating system (set for Windows and export for UNIX or LINUX
   operating systems) and JVM vendor as follows:

   `ANT_OPTS=-Xmx768m -Xms768m -XX:MaxPermSize=128m` for Sun and
   HP JVMs

   `ANT_OPTS=-Xmx768m -Xms768m` for IBM and JRockit JVMs

5. From the `<YFS_HOME>` directory, create the database JAR file using the
   following command applicable for your application server:

   — For WebLogic, run `ant -f bin/buildWLS.xml db-extn`

   — For WebSphere, run `ant -f bin/buildWS.xml db-extn`

   where `<YFS_HOME>` refers to the `<YANTRA_HOME>/Runtime` directory.

   This creates a `yfsdbextn.jar` file in the `<YFS_HOME>/extn/` directory.

   > **Note:** If you are running Sterling Supply Chain
   > Applications on SQL Server, create the database JAR file
   > using the following command:
   >
   > For WebLogic, run
   >
   > `ant -f bin/buildWLS.xml db-extn-sqlserver`
   >
   > For WebSphere, run
   >
   > `ant -f bin/buildWS.xml db-extn-sqlserver`

6. From the `<YFS_HOME>` directory, create the EAR file using the
   following command applicable for your application server:

   — For WebLogic, run `ant -f bin/buildWLS.xml create-ear`

   — For WebSphere, run `ant -f bin/buildWS.xml create-ear`

   where `<YFS_HOME>` refers to the `<YANTRA_HOME>/Runtime` directory.

   Instead of creating the EAR after extending the database, you can
   choose to update the EJBs and EAR. For more information on the

various targets for the building the EAR see the *Sterling Supply Chain Applications Installation Guide*.

7. Modify the `agentserver.sh` script (or `.cmd` file on Windows) to include the `<YFS_HOME>/extn/yfsdbextn.jar` file in the beginning the CLASSPATH environment variable. For more details about this script, see the *Sterling Supply Chain Applications Installation Guide*.

8. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

9. Deploy the new `yantra.ear` file according to your application server documentation.

If you want to include your custom or extended entity XMLs into the ERD descriptions, see Section 7.7, "Merging Documentation Folders" for more information on merging.

7.6 Custom Code Requirements to Avoid Deadlocks

Deadlock refers to a specific condition in a database, when two processes are waiting for the other process to release a resource. For example when one client application holds a lock on a table and attempts to obtain the lock on a second table that is held by another client application, this may lead to a deadlock if the other application attempts to obtain the lock that is held by the first application.

To circumvent the deadlock problem, Sterling Commerce recommends that you need to sort the information to be accessed in a certain order before grabbing locks. This is applicable to situations where you need to grab multiple inventory item locks within a single transaction boundary. However, you do not need to sort if you call the APIs to process single items per transaction commit.

Reading Uncommitted data in DB2 Database
In DB2, when you select a record from a table, a read lock is obtained on the record. If the record being selected has been updated but not committed, the thread will wait until it commits the changes. Alternatively you could read the record with Uncommitted Read (UR) in

which case the latest value that has been updated is provided to the user.

You can read uncommitted data from any list API by enabling the `ReadUnCommitted` attribute to `Y` in its input XML. As a result the locking scenario is circumvented in DB2 database.

This behavior is different from Oracle, hence if you are writing custom code on DB2 you should understand this behavior to avoid lock escalations.

7.7 Merging Documentation Folders

The merging of foundation and PCA documents is now done as part of the creating Sterling Supply Chain Applications runtime process. The "merge" ant script for creating Sterling Supply Chain Applications runtime internally calls the "mergedocumentation" ant target. The "mergedocumentation" ant target will merge the documentation folder of PCAs with the foundation documentation folder to have a single source of information across applications. For more information about creating Sterling Supply Chain Applications Runtime, see the *Sterling Supply Chain Applications Installation Guide*.

The "mergedocumentation" ant target merges the following foundation and any PCA documents:

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/foundati on/online_help`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/foundati on/xmlstruct`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/foundati on/api_javadocs`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/foundati on/core_javadocs`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/foundati on/code_examples`

- `<<YANTRA_ HOME>/Applications/Foundation/documentation/source/<PCACode >/online_help`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/<PCACode >/xmlstruct`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/<PCACode >/api_javadocs`

- `<YANTRA_ HOME>/Applications/Foundation/documentation/source/<PCACode >/code_examples`

After the merging is complete, the merged documents are placed in the `<YFS_HOME>/documentation` directory.

8

Programming Transactions

You can extend Sterling Supply Chain Applications programmatically both to enhance the functionality of your implementation of Sterling Supply Chain Applications and to integrate with external systems.

This chapter describes how to achieve extensibility programmatically, using these mechanisms. For guidelines on using Sterling Supply Chain Applications naming conventions in your extensions, see Appendix A, "Special Characters Reference". Transaction processing mechanisms in Sterling Supply Chain Applications can be classified into two basic categories:

- Synchronous (on demand) or asynchronous (message driven) services

- Time-triggered transactions

8.1 Services

In Sterling Supply Chain Applications terminology, a service is core business logic component that is stateless and does not contain presentation logic. Each service (either provided out-of-the-box by Sterling Supply Chain Applications or those that are custom created using the Sterling Service Definition Framework) represents a logical unit of processing that can be independently performed without any loss of data integrity and within one transaction boundary. Using the Sterling Service Definition Framework, one or more services can be aggregated into larger composite services which can in turn be used to create other services. This provides a way to build small reusable components that can be linked together to provide complex business processing.

All services within the Sterling Service Definition Framework can be invoked bi-directionally—either through internal Sterling Supply Chain Applications business processes or through external systems. Services deployed in the Sterling Service Definition Framework are stateless, each having their own transaction commitment boundaries.

A service can be invoked by Sterling Supply Chain Applications by associating the service with an event through an action. You can use a standard interoperability event handler or implement your own custom event handler. You can then configure Sterling Supply Chain Applications to invoke the event handlers when certain events are raised and conditions are met. For more information about configuring events, conditions, and actions, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

Sterling Supply Chain Applications provide several user exits to extend business logic. User exits invoked from within transactions can be associated to a service when configuring transactions. Note that templates are not supported for user exits. For more information on configuring user exits, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

Once services have been configured, they can also be invoked programmatically by a client.

The service invocation configuration depends on the location of the client invoking the service in relation to the location of the Sterling Supply Chain Applications installation, as described in the following situations:

- If the invoking client **do not** have Sterling Supply Chain Applications installed - configure for remote invocation.
- If the invoking client **do** have Sterling Supply Chain Applications installed - configure for local invocation.

8.1.1 Method of Invocation

Depending upon the mode of invocation, services can be classified into two major categories:

- Synchronously invoked (on demand) - These services can perform all their processing and return the result in single call.
- Asynchronous (message driven)

Synchronously invoked

These services can perform all their processing and return the result in a single call, on demand.

Asynchronous

These services automatically perform all their processing whenever triggered by a message from an external system or from within Sterling Supply Chain Applications. The trigger could be in the form of a file, a database record or a message in a message queue depending upon the mode of integration. These services do not return any value and are purely used for background processing such as sending out emails or automatically receiving updates from or sending updates to an external system.

In general, asynchronous services provide a lower cost to performance ratio than synchronous services and should be preferred wherever possible. However, asynchronous services queue up and process messages in the order they are received. The time to process a certain transaction after it's been queued can vary widely depending upon peaks in your processing cycle and a host of other factors. Therefore, they are not suitable for certain specific scenarios where an SLA (service level agreement) requires that a transaction has to be processed within a specified short time frame. However, these scenarios are rare for most businesses and business processes and asynchronous processing is efficient enough for the majority of transactions at a significantly lower cost while still providing a high service level.

8.1.2 The Business Function Library

A service typically consists of one or more messaging components (or components that define how messages to and from the service are handled), one or more utility components (such as email or alert handlers) and one or more business processing components. For information about the utility and messaging components available for services defined in Sterling Supply Chain Applications, see the *Sterling Supply Chain Applications Platform Configuration Guide*. This section describes how to work with service, customize and extending the business processing components, and make them usable in services.

Sterling Supply Chain Applications are shipped with an extensive out-of-the-box business function library. Each function in this library is known as a standard API. For detailed information on the input, output,

and behavior of each standard API, see the *Sterling Supply Chain Applications Javadocs*.

This chapter discusses common programming, customization and extensibility patterns that are applicable to large portions of the Sterling Supply Chain Applications business function library.

You can also write your own business functions and use them in services. Each such function is known as an extended API.

While standard APIs can be aggregated and linked together to form more complex services, for most cases the API provides all the functionality that is required for a business transaction and is therefore not required to be linked together with other components or APIs. To ease working with this most common scenario, all of the standard APIs are automatically available for synchronous invocation without the need to model each one as a service using the Sterling Service Definition Framework. You can think of these APIs as "automatically defined" synchronous services. However, extended APIs and asynchronous invocation of these APIs requires that you explicitly model them as services first using the Sterling Service Definition Framework.

8.1.3 Message Size

If a database table reaches its maximum size and a `send()` function attempts to insert a message in the table, the Sterling Service Definition Framework throws an exception. See the size limitations noted in the Table 8–1:

Table 8–1   Message Size for Asychronous Transports

Mechanism	Data Type	Size
Database (Oracle)	LONG	2 GB
Database (SQL Server)	TEXT	4 GB
JMS Queue	TextMessage	4 GB
MSMQ Queue	Message	2 GB

8.1.4 Exception Handling

The Alert Console displays all exceptions logged by the Sterling Service Definition Framework. It also enables you to reprocess exceptions that

occur in transactions configured to be asynchronous. When using a database or queue, calls are asynchronous.

The Sterling Service Definition Framework uses the log4j utility for logging exception information. The log4j utility writes both trace and debug information to a log file. You can configure the logger to send different categories of messages to different destinations. Categories are organized hierarchically, which permits inheritance. Each category can be configured with a priority indicating a severity level. If a category is not configured with a priority, it inherits the priority of its closest ancestor with an assigned priority.

All exceptions that occur during an API call or during use of an event handler are logged.

8.2 APIs

You can use both standard APIs that are supplied by Sterling Supply Chain Applications and any extended (custom) APIs that you have created. Sterling Supply Chain Applications provide standard APIs to handle the most common business scenarios. For example, there are APIs that create an order, allocate an order, and report shipment confirmation. Standard APIs can be invoked directly or aggregated into more complex services.

8.2.1 API Behavior

Each API takes an XML document as input and returns another XML document as output. The `YFSEnvironment` input parameter represents a runtime state under which this API is being invoked. It is used by Sterling Supply Chain Applications for the following tasks:

- Security audits and logging

- Transaction control

- Achieving invocation-specific API behavior

For an asynchronous service, Sterling Supply Chain Applications automatically create an instance of this object and passes it to each API part of the service. To programmatically invoke a synchronous service, you have to create an instance of this environment by calling the `createEnvironment()` API.

> **Note:** In general, input to APIs should not contain any "BLANK" elements or attributes. A blank element can be defined as an element containing all the attributes with blank values. If a blank element is passed, the API behavior is unpredictable.

All APIs (whether standard or extended) have the same signature with respect to input parameters and return values. This signature is of the form

```
org.w3c.dom.Document APIName(YFSEnvironment env, org.w3c.dom.Document input);
```

In order for custom APIs to access custom values, the API should implement the `com.yantra.interop.japi.YIFCustomApi` interface. If entered, these name/value pairs are passed to the Custom API as a Properties object. See the *Sterling Supply Chain Applications Javadocs* for more information about the `com.yantra.interop.japi.YIFCustomApi` interface.

8.2.2 Types of APIs

An API processes records based on key attribute values, processing records with a primary key first. If the primary key is not found, the API then searches for the logical keys and then processes those records. For example, the `ChangeOrder()` API first looks for the OrderHeaderKey key attribute and then for the combination of the OrderNo and EnterpriseCode key attributes.

The APIs of Sterling Supply Chain Applications can be classified as one of the following types:

- Select APIs
- List APIs
- Update APIs

Select APIs

Typically prefixed with `get`, select APIs return one record for an entity (for example, the `getOrderDetails()` API returns the details of one order). They do not update the database.

Since select APIs return only one record, they require unique key attributes to be passed in the input XML. If a unique key attribute is not passed in the input XML, the API uses blanks for those attributes in the criteria to select the record. There can be more than one unique key combination, and in that combination you must pass any one of the multiple combinations.

For example, an order is uniquely identified either by the `OrderHeaderKey` key attribute or by a combination of the `OrderNo` and `EnterpriseCode` attributes. So, when calling the `getOrderDetails()` API, you must pass either the `OrderHeaderKey` attribute or the combination of the `OrderNo`, `EnterpriseCode` and `DocumentType` key attributes. If you pass only `OrderNo`, the API returns the order that matches `OrderNo` and has a blank enterprise code. In order to identify the unique key combinations for each API, see the *Sterling Supply Chain Applications Javadocs*.

However, `getOrderDetails()` API uses a select for update on `YFS_ ORDER_HEADER` so that its internal processes such as user exits, events, etc., have a lock on the order elements while the thread working on it is active. This enables to maintain a transaction cache until the final commit. Hence, you need to avoid using nested transactions to overcome the locking mechanism by performing:

1. Commit or rollback only once for all event of the order. Keep in mind, that all the events are set to rollback if one of them fails.

2. Select the order for each event and process. Also keep in mind, that if age of the orders having multiple events are higher it could have an impact on the performance.

List APIs

Typically prefixed with `get`, list APIs return a list of records for an entity that match the criteria specified through the input XML (for example, the `getOrderList()` API returns a list of orders). For more information on specifying search criteria, see . If any attribute in the input XML has a blank value, it is ignored. List APIs do not update the database.

Update APIs

Update APIs insert new records into the database. They also modify or delete existing records in the database. Update APIs that modify or

delete existing records use the same logic as select APIs to identify which record to update. If no record is found, update APIs throw an exception.

8.2.3 Date-Time Handling

Sterling Supply Chain Applications handle values for both date-time and date. Date-time refers to values that contains a date and time component, where Date refers to values that contain only a date component.

Date values can be made nullable by specifying `Nullable="true"` in the entity XML. Thereby the Date values in the table is blanked out. The expected behavior of a date column is marked as Y is described in Table 8–2.

Table 8–2   Nullable Date Behavior

Action	Description
Insert	When the field is not populated in a database object (is null), the database infrastructure automatically inserts a null value into the column in the database.
Update	When the application nulls out a date, it sets the corresponding field to null value in the database.
Select or List	When a column is defined as nullable and the date from the database is returned as null, it is automatically nulled out. So, the corresponding get method returns a null.
Search by date	Can pass null value as needed when specified to do so.

> **Note:** If you have specified the date value as 01/01/2400
> in versions prior to Release 7.9, those values are now
> treated as null. The dates with special significance are:
>
> - Null date - 01/01/2400
>
> - High date - 01/01/2500
>
> - Low date - 01/01/1900

8.2.3.1 Specifying Time Zones

Dates and times are time zone aware. Time zones are relative to the
Coordinated Universal Time (UTC).

For example, if an order is created on the system on 06/15/2003 at
16:00:00 in New York, (USA/New York time zone) a user in Chicago who
examines that the order will see that the order creation date-time as
06/15/2003 at 15:00:00, (USA/Chicago time zone).

For a time published from Boston that is -5:00 hours from UTC, the
string literal "-5:00" is appended to the current date-time attribute
published from APIs. The input "2003-04-23T14:15:32-05:00" gives the
date, time, and time zone reference for a transaction.

The Sterling Supply Chain Applications time zone is specified by the
`yfs.install.localecode` parameter in the `yfs.properties` file (for
example, `yfs.install.localecode=EN_US_EST`).

8.2.3.2 Using Date-Time Syntax

All APIs, user exits, and events that use date-time fields have a uniform
syntax (a combination of the basic and extended formats of the ISO
8601 specification). This syntax is the expected format for all input as
well as output. For details on the values this format uses, see
Section C.4.8, "getDateValue" on page 603.

Date Only Syntax
YYYY-MM-DD

Date-Time Syntax

YYYY-MM-DDTHH**:**MI**:**SS$\underline{+}$HH**:**MM

Values in bold are placeholders for literals. For example, the format for March 5, 2003, 11:30:59 p.m. is 2003-03-05T23:30:59.

Syntax Parameters

YYYY - Required. Four-digit year. Used in both date-time and date fields.

MM - Required. Two-digit month. Used in both date-time and date fields.

DD - Required. Two-digit day of the month. Used in both date-time and date fields.

T - Required. The literal value T, which separates the date and time component. Used only in date-time fields.

HH - Required. Two-digit hour of the day. For example, 11 p.m. is displayed as 23:00:00. Used only in date-time fields.

MI - Required. Two-digit minutes of the hour. For example, 59 minutes is displayed as 00:59:00. Used only in date-time fields.

SS - Required. Two-digit seconds of the minute. For example, 21 seconds is displayed as 00:00:21. Used only in date-time fields.

$\underline{+}$**HH:MI** - Optional. Two-digit hours and minutes, separated by a colon (":"). Indicates how many hours from UTC, using - to indicate earlier than UTC and + to indicate later than UTC. If this value is not passed in input, the time zone of the Sterling Supply Chain Applications are assumed.

8.2.4 API Input XML Files

APIs retrieve data using input XML files that define which records need to be selected or used. When extending the database to include additional fields, you need to also extend the input XML to populate those fields.

> **Caution:** Do not pass a blank element (an element containing all the attributes with blank values) to an API. Also, do not pass attributes that have leading or trailing spaces. The result of either situation is not predictable.

Example 8–1 shows an input XML modification.

Example 8–1   Example of Input XML Modification

The following example modifies the input XML file for the YFS_ createOrder() API:

```
<Orders AuthenticationKey="">
   <Order EnterpriseCode="DEFAULT" OrderNo="DB04" OrderName="DB04"
OrderDate="20010803" OrderType="Phone" PriorityCode="1" PriorityNumber="1"
ReqDeliveryDate="20010810" ReqCancelDate="" ReqShipDate="20010810"
SCAC="FEDEX" CarrierServiceCode="Express Saver Pak"
CarrierAccountNo="112255" NotifyAfterShipmentFlag="N" NotificationType="FAX"
NotificationReference="" ShipCompleteFlag="N" EnteredBy="Iain "
ChargeActualFreightFlag="Y" AORFlag="Y" SearchCriteria1="Search"
SearchCriteria2="Search Again" >
      <OrderLines>
         <OrderLine PrimeLineNo="1" SubLineNo="1" OrderedQty="1"
ReqDeliveryDate="20010810" ReqCancelDate="20010810" ReqShipDate="20010810"
SCAC="FEDEX" CarrierServiceCode="Express Saver Pak" PickableFlag="Y"
HoldFlag="N" CustomerPONo="11" >
            <Extn ExtnAcmeLineType="Type1"/>
            <Item ItemID="ITEM1" ProductClass="A" ItemWeight="1"
ItemDesc="paintball gun" ItemShortDesc="pball gun" UnitOfMeasure="EACH"
CustomerItem="Spectra Flex" CustomerItemDesc="GEGRG" SupplierItem="Spectra
Flex @ supplier" SupplierItemDesc="Spectra Flex Desc @ supplier"
UnitCost="15.99"  CountryOfOrigin="CA"/>
            <PersonInfoShipTo Title="Mr" FirstName="Quigley" MiddleName="Al"
LastName="Johns" Company="Company" JobTitle="Project Clert"
AddressLine1="Address Line 1 -3 Main Street" AddressLine2="ShipTo Address
line 2" AddressLine3="ShipTo Address line 3" AddressLine4="ShipTo Address
line 4" AddressLine5="ShipTo Address line 5" AddressLine6="ShipTo Address
line 6" City="Acton" State="MA" ZipCode="01720" Country="US"
DayPhone="978-635-9242" EveningPhone="978-635-9252"
MobilePhone="978-888-8888" Beeper="" OtherPhone="other555-5555" DayFaxNo=""
EveningFaxNo="" EMailID="jquigley@maine.com"
AlternateEmailID="hfournier@ontario.com" ShipToID=""/>
         </OrderLine>
```

```
      <NumberOfOrderLines/>
    </OrderLines>
    <PersonInfoShipTo Title="MR" FirstName="s" MiddleName="X" LastName="T"
Suffix="T" Department="T" Company="SD" JobTitle="SS" AddressLine1="SS"
AddressLine2="SS" AddressLine3="SS" AddressLine4="SS" AddressLine5="SS"
AddressLine6="SS" City="REDWOOD" State="CA" ZipCode="01852" Country="USA"
DayPhone="3456789234" EveningPhone="3456789234" MobilePhone=""
EveningFaxNo="SS" />
      <PersonInfoBillTo Title="mj" FirstName="m" MiddleName="JJ"
LastName="KK" Suffix="lll" Department="l" Company="kj" JobTitle="k"
AddressLine1="HJHKK" AddressLine2="HJKHK" AddressLine3="HKHJ"
AddressLine4="" AddressLine5="" AddressLine6="" City="UUU" State="IUI"
ZipCode="78787" Country="USA"  />
  </Order>
  <NumberOfOrders/>
</Orders>
```

> **Important:** In order for the factory setup scripts to
> operate properly, when you add a column to a database
> table, be sure that the column is not null and that it has a
> default value. If you need to make the column nullable, the
> default value must not be present.
>
> Also, when you are specifying XML Name and XML Group,
> keep in mind that the values should be valid Document
> Object Model (DOM) strings. (The values must not contain
> spaces or special characters that are not supported by the
> DOM specification.)

The following example XML file adds a column to the YFS_ORDER_LINE
table:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<DBSchema>
  <Entities>
    <Entity TableName="YFS_ORDER_LINE">
      <Attributes>
        <Attribute ColumnName="EXTN_ACME_LINE_TYPE" DecimalDigits="" Default
         Value="' '" Size="10" Type="CHAR" XMLGroup="Extn"
         XMLName="ExtnAcmeLineType"/>
      </Attributes>
    </Entity>
  </Entities>
```

</DBSchema>

8.2.5 Forming Queries in the Input XML of List APIs

The input XML of list APIs enable queries on conditions such as *starts with*, *contains*, *is greater than*, and so forth. Example 8–2 shows a fragment of the input XML that returns a list of items at a specific shipping node that fall within a specific weight range and to be shipped during a specific date range.

Example 8–2   getOrderList API Input XML with Query Type Values

```
<Order ReqShipDateQryType="DATERANGE" FromReqShipDate="20010113"
ToReqShipDate="20030113" /Order>
<OrderLine ShipNode="Atlantic" /OrderLine>
<Item ItemWeightQryType="BETWEEN" FromItemWeight="2" ToItemWeight="20"/>
<OrderRelease CarrierServiceCodeQryType="FLIKE" CarrierServiceCode="Priority" />
```

To form queries:

1. Edit the custom input XML of any list API, and append `QryType` to any attribute you want to query on. Any attribute that is not appended with `QryType` can also be queried on, using the default query type value EQ, as shown for `ShipNode` in Example 8–2.

2. For attributes appended with `QryType`, specify a query type value from Table 8–3. This is case sensitive.

3. Specify the values that are applicable to your search criteria.

The values for the `QryType` attributes vary depending on the datatype of the field. Table 8–3 lists the supported query type values for each datatype.

Table 8–3   Query Type Values Used by List APIs

Field DataType	Supported Query Type Values
Char/VarChar2	• EQ - Equal to
	• FLIKE - Starts with
	• LIKE - Contains
	• GT - Greater than
	• LT - Less than

Table 8–3   Query Type Values Used by List APIs

Field DataType	Supported Query Type Values
Number	• BETWEEN - Range of values
	• EQ - Equal to
	• GE - Greater than or equal to
	• GT - Greater than
	• LE - Less than or equal to
	• LT - Less than
	• NE - Not equal to
Date	• DATERANGE - Range of dates
	• EQ - Equals
	• GE - Greater than or equal to
	• GT - Greater than
	• LE - Less than or equal to
	• LT - Less than
	• NE - Not equal to
Date-Time	• BETWEEN - Range of dates
	• EQ - Equals
	• GE - Greater than or equal to
	• GT - Greater than
	• LE - Less than or equal to
	• LT - Less than
	• NE - Not equal to
Null	• ISNULL - Return records that are null.
	• NOTNULL - Return records that are not null.
	Note: These two query types are used when the column or attribute is set to Nullable in the entity XML.

8.2.6 API Output XML Files

APIs return data using two types of output XML files that define which elements and attributes are required by an API.

- Output XML File - Defines the outer limits of the data an API can return. Do **not** modify output XML files.

- Template XML File - Defines the data returned by an API for the record specified in the input XML file and restricts the amount of data to a subset of the output XML. You can modify this file to incorporate a subset of the attributes and elements from the output XML.

8.2.7 Output XML Templates

Many APIs use a corresponding output template. The output template is in XML format and is read in by an API in order to determine the elements and attributes for which it should return. The standard output template defines the elements and attributes returned for any specific API. (To see the entire range of possible values an API can return, see its output XML in *Sterling Supply Chain Applications Javadocs*.) The standard template can be a subset of the entire range of values returned, as determined by the output XML in the *Sterling Supply Chain Applications Javadocs*.

> **Note:** Ensure that when adding elements and attributes to the output template, use *only* those that are documented in the *Sterling Supply Chain Applications Javadocs*. While the APIs can output additional elements and attributes, only those that are documented in the *Sterling Supply Chain Applications Javadocs* are supported.

For example, the standard output template of the `getOrderList()` API returns the header-level information of an order and the standard output template of the `getOrderDetails()` API returns in depth information about an order.

Besides the standard output XML template, you can create custom output templates for APIs to use for your own business requirements, such as different output for different document types.

Document Types

If you use a variety of business-related document types such as orders, planned orders, purchase orders, and returns, you can use custom templates that enable an API to return the values that pertain to each unique document type.

For example, you can use one template with the `getOrderDetails()` API to return information about Planned Orders and another template for the `getOrderDetails()` API to return different information about Orders.

Standard Output Template Behavior

The set of values that the standard output template returns covers a variety of business scenarios. With such a large range of possibilities, an API using the standard output template may return much more data than you need for your business purposes (and take much more time to process than you prefer).

If you want to customize the information returned by an API, you can do so by creating and using a custom template, using our guidelines and procedures.

8.2.8 Extending an Output XML Template

Many APIs use an output XML template to define what is returned. Each API has its own XML template, which is picked up from the `<YANTRA_HOME>/Applications/Foundation/template/api/<apiName>.xml` file. The files in this directory are part of the product and should not be altered. However, these templates can be overridden by implementing template extensions.

To extend a template file:

1. Copy the template `<YANTRA_HOME>/Applications/Foundation/template/api/<apiName>.xml` file, to the `<YANTRA_HOME>/Applications/Foundation/template/api/extn/` directory, keeping the same file name.

2. Modify the copied file, as needed. To extend a template file, add the `Extn` tag under the entity tag. For example, if you have added a column `EXTN_COLOR` to `YFS_ITEM` table, you also must add the tag `Extn` under the tag `Item` in the `getItemDetails.xml` file as follows:

```
<Item ItemKey=""....>
    <PrimaryInfo MasterCatalogID="" .../>
    ...
    <Extn ExtnColor=""/>
</Item>
```

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

8.2.9 Best Practices for Creating Custom Output XML Templates

Whenever you call an API, you need to pass your own customized template, not the sample provided by Sterling Supply Chain Applications. This section helps guide your decision-making processes in planning how to design custom output templates.

Gather Information Relevant to the API

Custom output templates provide the flexibility to return whatever data you wish, so it is important to understand that it is possible to modify an output template in such a way that it returns information that is not quite relevant to the API.

For example, it is possible to modify the output template of the `getOrderList()` API in such a way that it returns detailed information about an order rather than just header-level information. You should modify an output template in such a way that it takes advantage of the unique aspects of its corresponding API. Keeping each template unique to its API prevents any ambiguity about which API to use in any specific situation.

Gather Information Relevant to Your Business Needs

Since the standard output template returns all attributes, even for empty elements in the template, you might want to tailor information to your specific business needs. If you don't exclude the attributes you don't require, you receive more data than you need and the extra data may slow the performance of the API.

For example, if you are using the `getOrderDetails()`API to return only `OrderLine` attributes but your custom output template includes `Schedule` attributes, all attributes for `OrderLine` and `Schedule` are returned.

Choose an Appropriate Template Mechanism

In general, the format of any template should follow the same structure as the standard template. Keeping this general rule in mind, there are two ways to customize the standard template, differentiated by the amount of data they return and how they can be called:

- Static templates
- Dynamic templates

Static templates provide the ability to add new elements but not remove any of the defaults. A static template is pervasive, as it is picked up by default by an API whenever that API is invoked.

Dynamic templates provide the ability to add new elements and remove any of the default elements from the standard template. A dynamic template is an instance, as it is picked up only for a specific API call, such as when configured to do so during user interface extensibility.

A comparison of the differences between the two types of template mechanisms is summarized in Table 8–4.

Table 8–4   Comparison of API Output Template Mechanisms

Template Types	Allowed XML Elements	Behavior
Static Template	Default template elements cannot be removed. New elements can be added.	Pervasive. Picked up by default by an API.
Dynamic Template	Default template elements can be removed. New elements can be added.	Instance. Picked for a specific API call, as configured during user interface extensibility.

Choose which of these mechanisms best fits your business needs and adhere to it.

Remember that when you define a dynamic template, all possible values are returned. In order to return the smallest amount of data for an element, when you are pruning away elements you don't need, you need to include its parent with at least one of its attributes.

If you leave an element blank or include unwanted attributes in the parent element all values are returned, as illustrated in Example 8–3.

Example 8–3   A Poorly Pruned Dynamic Template

```
<!-- getOrderDetails Output XML -->
<Order>
    <OrderLines>
    <!--1 or more order line-->
        <OrderLine>
            <Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
            <Schedules>
            <Schedule Attr1 ...... />
            </Schedules>
        </OrderLine>
    </OrderLines>
<Order>
```

Since Example 8–3 specifies all OrderLine attributes as well as a few Item and Schedule attributes, the API returns values similar to those in Example 8–4.

Example 8–4   Values Returned by a Poorly Pruned Dynamic Template

```
<OrderLine AllocationDate="03/28/2002" CarrierAccountNo="112233"
CarrierServiceCode="Next Day Air" Createprogid="SterlingTester"
Createts="03/28/2002" Createuserid="SterlingTester" CustomerLinePONo="999"
CustomerPONo="111" DeliveryCode="AIR" DepartmentCode="Clothing" ExtendedFlag=""
ExternalReference1="" ExternalReference2="" ExternalReference3=""
ExternalReference4="" ExternalReference5="" FreightTerms="Buyer" HoldFlag="N"
HoldReasonCode="HoldReas" ImportLicenseExpDate="08/08/2002"
ImportLicenseNo="225588" InternalReference1="" InternalReference2=""
InternalReference3="" InternalReference4="" InternalReference5="" KitCode=""
LineClass="" LineSeqNo="1.1" LineType="Single" Lockid="1" MarkForKey=""
Modifyprogid="SterlingTester" Modifyts="03/28/2002"
Modifyuserid="SterlingTester" OrderClass="NEW"
OrderHeaderKey="200203281036245174" OrderLineKey="200203281036245175"
OrderedQty="5.00" OrigOrderLineKey="" OriginalOrderedQty="5.00"
OtherCharges="0.00" OtherChargesPerLine="0.00" OtherChargesPerUnit="0.00"
PackListType="Bill" PersonalizeCode="PersCode" PersonalizeFlag=""
PickableFlag="Y" PricingDate="01/01/2500" PrimeLineNo="1" Purpose="Purpose"
ReceivingNode="B1N1" ReqCancelDate="01/01/2500" ReqDeliveryDate="04/04/2002"
ReqShipDate="03/30/2002" ReservationID="" ReservationPool="" SCAC="UPS"
ShipNode="E1N1" ShipToID="" ShipToKey="" ShipTogetherNo="Y" SplitQty="0.00"
SubLineNo="1" TotalDiscountAmount="0.00" TotalOtherCharges="0.00">
<Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
<Schedules>
<Schedule ExpectedDeliveryDate="" ExpectedShipmentDate="" TagNumber=""
OrderHeaderKey="" OrderLineKey="" OrderLineScheduleKey="" ScheduleNo=""
```

```
ShipByDate="" Quantity=""  PromisedApptStartDate="" PromisedApptEndDate=""/>
</Schedules>
</OrderLine>
</OrderLines>
</Order>
```

In Example 8–5, the dynamic template has been trimmed down, keeping in mind the following guidelines:

- The structure of the custom output template mirrors the structure of the standard output template.

- Excess elements (regarding kits, schedules, addresses, and so forth) are pruned away.

- Parent elements are populated with one attribute in order to suppress excess detail. For example, specifying the OrderNo attribute for the Order element suppresses all of the other Order attributes.

Example 8–5   A Carefully Pruned Custom Output Template

```
<!-- getOrderDetails Output XML -->
<Order OrderNo="">
    <OrderLines>
        <!--1 or more order line-->
        <OrderLine PrimeLineNo="">
            <Item CountryOfOrigin="" ItemDesc="" ItemID=""/>
        </OrderLine>
    </OrderLines>
</Order>
```

Since Example 8–5, "A Carefully Pruned Custom Output Template" specifies only a few Item attributes, and only one attribute for its parent element, the getOrderDetails() API returns only the values shown in Example 8–6.

Example 8–6   Values Returned by a Carefully Pruned Output Template

```
<?xml version="1.0" encoding="UTF-8" ?>
<Order OrderNo=Y00000765>
    <OrderLines>
        <OrderLine PrimeLineNo="1">
            <Item CountryOfOrigin="IN" Item Description" ItemDesc="Green Sari"
            ItemID="GNSARI5LT" />
        </OrderLine>
```

```
        <OrderLine PrimeLineNo="3">
            <Item CountryOfOrigin="CA" ItemDesc="Pink Scarf" ItemID="PKSCARF4LT"
            />
        </OrderLine>
    </OrderLines>
</Order>
```

This method of pruning the templates will improve the performance as database access to order schedules and other unwanted elements has been prevented.

Develop Useful Templates

The supplied templates located in the `<YANTRA_ HOME>/Applications/Foundation/template/api/` directory are sample guides. Use them to help you develop your own output XML templates. Using your own customized templates gives you much more flexibility, greater performance, and more assurance of appropriate data output. You can either pass your template through the env or put it in the extension folder.

Keep Performance Needs in Mind

Besides tailoring the templates to your business needs, it is important to keep technological considerations in mind. For performance-related information about using output, see the *Sterling Supply Chain Applications Performance Management Guide*.

8.2.10 Customizing an Output Template

In general, when you customize an output template, you do so by editing a copy of the standard template. You cannot modify the standard output template.

There are two ways of customizing and calling the output template. Which function you choose depends on the size and type of the data set you want returned by the API.

If you want a set of data that is *equal to or greater than* the standard output template contains, see Section 8.2.11, "Defining and Deploying a Static Template" on page 476. If you want a set of data that is *less than or greater than* that which the standard output template contains, see Section 8.2.12, "Defining and Deploying a Dynamic Template" on page 477.

8.2.11 Defining and Deploying a Static Template

If you want to use a template that has more elements in addition to those in the standard output template, create a static output template. This function enables you to create a template that includes all of the elements in the standard output template *plus* any new ones you add. For example, you may need to add UI fields for any database columns you have added. Note that if you use this function, you *cannot* remove any elements that exist in the standard template.

To define and deploy a static template:

1. Copy the standard output template for the API that you want to modify from the `<YANTRA_ HOME>/Applications/Foundation/template/api/<FileName>.xml` file to `<YANTRA_ HOME>/Applications/Foundation/template/api/extn/<FileName>[ .<DocType>].xml`.

   - Keep the file name of your new template the same as the standard template.

     The name of the output template corresponds with the name of the API or event associated with it. For example, the `getOrderDetails()` API takes the output template file `getOrderDetails.xml`.

   - If the template references a document type, include the document type code in the filename.

     For example, to create an output template for the `getOrderDetails()` API for an Order (`0001`) document type, the name of the template XML is `getOrderDetails.0001.xml`.

2. Modify the copied template in the `/template/api/extn/` directory as required, keeping in mind the guidelines described in Section 8.2.9, "Best Practices for Creating Custom Output XML Templates" on page 471.

   > **Note:** You may add any elements you wish, but you cannot remove any of the elements present in the standard output template.

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4. Call the API as typical and it automatically picks up the custom output template from the directory containing the custom templates.

8.2.12 Defining and Deploying a Dynamic Template

If you want to use a template that contains a subset of the elements in the standard output template, create a dynamic output template. If you want the ability to remove some elements from the standard template and perhaps add your own elements, you do that by passing your XML data or a file name into the YFSEnvironment object.

To define and deploy a dynamic template:

1. Copy the standard output template for the API that you want to modify from the `<YANTRA_ HOME>/Applications/Foundation/template/api/<FileName>.xml` file to `<YANTRA_ HOME>/Applications/Foundation/template/api/extn/extn_ <FileName>.xml`.

   When naming your new template file, use the same name as the standard template and you *must* prefix it with `extn_` to indicate that it is an extension.

   The name of the output template corresponds with the name of the API or event associated with it. For example, the `getOrderDetails()` API takes the output template file `getOrderDetails.xml`.

2. Modify the copied template in the `/template/api/extn/` directory as required, keeping in mind the guidelines described in Section 8.2.9, "Best Practices for Creating Custom Output XML Templates" on page 471.

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4. During user interface extensibility, call the `setApiTemplate()` function on the YFSEnvironment object. This enables you to specify an output template before calling an API, using one of the following functions:

– XML *data* as a variable - as in the following example:

```
YFSEnvironment env = createEnv();
Document doc = getTemplateDocument();
env.setApiTemplate("getOrderDetails", doc);
private YFSEnvironment createEnv() {
//create new environment by passing the user id, program id, etc.
}
private Document getTemplateDocument() {
//create a Document object containing the desired template XML.
}
```

– XML *file* as a variable - as in the following example:

```
YFSEnvironment env = createEnv();
env.clearApiTemplates();
env.setApiTemplate("getOrderDetails", "extn_myOrderDetails.xml");
private YFSEnvironment createEnv() {
//create new environment by passing the user id, program id, etc.
}
```

The API then uses the template passed in through YFSEnvironment to produce the output XML document. For details about the YFSEnvironment interface, see the *Sterling Supply Chain Applications Javadocs*.

8.2.13 Understanding the Output XML Templates

Since Sterling Supply Chain Applications enable you to define multiple types of templates, in addition to the standard templates that you cannot modify, it is important to understand the order of precedence Sterling Supply Chain Applications implement when reading templates. This section describes how Sterling Supply Chain Applications uses the API and event templates.

8.2.13.1 API Templates

Table 8–5, "Output Template Order of Precedence" shows the sequence of precedence for determining which output template is used by an API. Events use a similar sequence of precedence, using the directories

described in Section 8.2.13.2, "Event Templates" on page 479. Note that templates are not supported for user exits.

Table 8–5   Output Template Order of Precedence

Priority	Output Template Path and File Name
1	setApiTemplate(<file>|<xmlDocument>) to YFSEnvironment
	When a file is specified, it is picked up from the <YANTRA_ HOME>/Applications/Foundation/template/api/extn/ directory.
2	<YANTRA_ HOME>/Applications/Foundation/template/api/extn/apiName.docType .xml
3	<YANTRA_ HOME>/Applications/Foundation/template/api/apiName.docType.xml. (Sterling Supply Chain Applications sample template; not for use in production.)
4	<YANTRA_ HOME>/Applications/Foundation/template/api/extn/apiName.xml
5	<YANTRA_ HOME>/Applications/Foundation/template/api/apiName.xml (Sterling Supply Chain Applications sample template; not for use in production.)

8.2.13.2 Event Templates

Event templates help determine what elements and attributes should be present in the output XML of an event. For events raised, see the relevant transactions in the *Sterling Supply Chain Applications Javadocs*.

To see which events take output templates, see the files in the `<YANTRA_ HOME>/Applications/Foundation/template/event/` directory. These templates can be overridden by files you place in the `<YANTRA_ HOME>/Applications/Foundation/template/event/extn/` directory.

The naming convention for templates is BaseTxnName.eventName.xml. For example, the on_success event of the `createOrder()` API uses the ORDER_CREATE.ON_SUCCESS.xml event template.

Note that templates are not supported for user exits.

8.2.14 DTD and XSD Generator

Every Sterling Supply Chain Applications API uses standard input, output, and error XMLs. These XMLs conform to the related Document Type Definition (DTD). For example, consider the following XML:

```
<?xml version="1.0" encoding="UTF-8">
<Order EnterpriseCode="DEFAULT" OrderNo="S100" />
```

The corresponding DTD for this XML is:

```
<!ELEMENT Order>
<!ATTLIST Order OrderNo CDATA #IMPLIED>
<!ATTLIST Order EnterpriseCode CDATA #REQUIRED>
```

To create such DTDs for the extended Sterling Supply Chain Applications XML, a tool called the `xsdGenerator.xml` is provided in the `<YFS_HOME>/bin` directory, where <YFS_HOME> refers to the `<YANTRA_HOME>/Runtime` directory. This tool converts a specially-formatted XML file into a DTD and XML schema definition (XSD). The command for running the tool is:

```
ant -f xsdGenerator.xml generate
```

You can also specify the following properties in the `<YFS_HOME>/bin/yantra.properties` file:

Table 8–6   XSD Generator Properties

Fields	Description
<YFS_HOME>	Required. Specify the path to the /Runtime folder in the <YANTRA_HOME> directory.
xsdgen.use.targetnamespace	Optional. The default value is Y. If set to Y, the XSD files are generated with a defined target namespace.
xsdgen.use.datatypeimport	Optional. The default value is Y. If set to Y, all the XSD files reference a single common XSD file containing all the common data type definitions. If set to N, each XSD file is created with a copy of the database definitions embedded within it.

The input XML files should be placed in the `<YFS_HOME>\documentation\extn\input` directory. The resulting DTD and XSD files are placed in the `<YFS_HOME>/documentation/extn/output/dtd`

and `<YFS_HOME>/documentation/extn/output/xsd` directories respectively.

Consider the following sample XML that can be placed in the input directory:

Example 8–7   Sample XML for Converting to an XSD and DTD

```
<Item yfc:DTDOccurrence="REQUIRED" ItemKey="" ItemID="REQUIRED"
OrganizationCode="REQUIRED" UnitOfMeasure="">
    <PrimaryInformation Description="" ItemType="" />
    <AdditionalAttributeList>
        <AdditionalAttribute Name="" Value=""/>
    </AdditionalAttributeList>
    <Extn ExtnAttr1="" ExtnRefId="">
        <CSTItemDataList yfc:DTDOccurrence="ZeroOrOne">
            <CSTItemData yfc:DTDOccurrence="ZeroOrMany" ItemDataKey=""
            Description="">
            <CSTItemExtraData yfc:DTDOccurrence="ZeroOrOne" CodeType=""
            DataType="" />
            <YFSCommonCode yfc:DTDOccurrence="REQUIRED" CodeName="" CodeType=""
            CodeValue="" />
            </CSTItemData>
        </CSTItemDataList>
    </Extn>
</Item>
```

Table 8–7   Special Attributes in your XML

Fields	Description
yfc:QryTypeSupported	This attribute determines whether or not the query type functionality is supported for the attributes in this element. If set to Y, it takes effect for all the elements.
yfc:ComplexQuerySupported	This attribute specifies whether or not a complex query type is supported. This attribute can only be present in the root element.
yfc:XSDType	The name of the type to use for the root element schema definition.

Table 8–7   Special Attributes in your XML

Fields	Description
yfc:DTDOccurrence	This attribute can contain any of the following values: • REQUIRED - This element must be present if the parent element is present. • ZeroOrOne - This element is optional, but may occur only once. • ZeroOrMany - This element is optional, but may occur multiple times. • OneOrMany - This element is required, and may occur multiple times.
xmlns	The namespace to use for the targetNameSpace in the output XSD. This attribute takes effect only if it is present in the root element.

The attributes with values of REQUIRED are generated as required attributes in the DTD and XSD. However, an existing required attribute cannot be marked as optional.

The attribute values can also be specified to supply additional constraints. A list of options is separated by a vertical bar (|). The value of the attribute must be one of the given options. This is only supported for data types based on strings. All the values are trimmed of the whitespace character unless the value itself is entirely spaces, in which case the enumerated option remains unchanged.

For example, SomeAttr="A | B | C | |" results in valid options of "A", "B", "C", " ", and "".

> **Note:**   The DTDs do not support enumerated values containing only whitespace characters. Therefore, restrictions of this type cannot be represented in the DTD.

The default input and output XMLs that can act as a base for your custom XML are located in the <YANTRA_ HOME>/Applications/Foundation/documentation/xmlstruct/

directory. Also note that the DTDOccurrence and REQUIRED data provided for the standard tables are inferred from the base file in the xmlstruct directory and do not need to be supplied. If they are provided, the existing information is overridden by any new information present in the custom XMLs. Any required datatype and relationship information are obtained from the entity XMLs.

> **Note:** Do not put your custom XMLs in the xmlstruct directory.

Therefore, when the tool is run these base XML files serve as a default to your custom XML files, which need only contain the changes made by you such as the extended elements and attributes. This allows future upgrades to safely modify the XML files in the xmlstruct directory. Re-running the XSD generation tool automatically picks up these updates.

The appropriate XML file in the xmlstruct directory associated with your custom XML is identified by the file name. Your custom XML may start with an optional prefix followed by an under-score and the base file name. For example, a custom XML file named Custom_File_YFS_getOrderDetails_input.xml refers to the YFS_getOrderDetails_input.xml file in the xmlstruct directory.

However, the naming convention is optional. For example, you can also name your custom XML sampleCustomApi.xml but no base file is used. In this case, the tool outputs an informational message to indicate that no base XML is found.

> **Note:** If you want to use our base XML file for conversion, the naming convention of your custom XML must be suffixed appropriately. For example, Custom_File_YFS_getOrderDetails_input.xml would use the base file named YFS_getOrderDetails_input.xml.

The generated XSD specifies the target namespace as shown below:

```
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://www.yantra.com/documentation"
```

```
xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:yfc="http://www.yantra.com/documentation">
```

This namespace is picked up from the `xmlns` attribute on the root element of the input XML and defaults to `http://www.yantra.com/documentation`.

The XSD and DTD files contain query type attributes used in list APIs when `QryTypeSupported="Y"` is set in the root element of the input XML. Similarly, the complex query types defined for `getItemList()` and `getOrganizationList()` APIs are represented in the XSD and DTD files when `ComplexQuerySupported="Y"` is set.

However, in APIs the following exceptions are exhibited in the DTDs since these constraints cannot be represented in a pure DTD, XSD or both:

- If an XML contains multiple `Extn` attributes, the generated DTD-only (not generated XSD) defines a single `Extn` element which appears as the union of all possible `Extn` elements.

- Conditionally required attributes. For example, you need to specify a group of attributes or another group of attributes such as `OrderHeaderKey` or `EnterpriseCode/OrderNo`.

- Mandatory condition of a node depends on some attribute value. For example in the `createOrder()` API, the `OrderLine` node is required if the `DraftOrderFlag="N"`.

8.2.15 Defining Complex Queries

Complex queries help to narrow a detailed listing obtained as output from an API. To generate the desired output, you can pass queries using `And` or `Or` operators in the input XML of an API.

For example, you can query the `getItemList` API based on the unit of measure, item group code or any parameters provided in the API definition, using the complex query operators, `And` or `Or`.

Complex queries are supported for the following APIs:

- getItemList
- getOrganizationList
- getOrderList
- getOrderLineList

- getShipmentList

  > **Note:** Only item, organization, order, order line, shipment
  > and shipment line entities are supported for performing
  > complex queries. The attributes for complex query must be
  > a valid database columns of these entities and should be at
  > the same level.
  >
  > For more information about these APIs, see *Sterling Supply
  > Chain Applications Javadocs*. For more information on valid
  > database columns, see Sterling Supply Chain Applications
  > ERDs.

Consider the following scenario for adding complex queries to the
`getItemList` API.

The `getItemList` API returns a list of items based on the selection
criteria specified in the input XML such as item attributes, aliases,
category, and so on. You can create complex queries in the
`getItemList` input XML as shown in Example 8–8.

Example 8–8   Adding Complex Queries in getItemList API

```
<Item OrganizationCode="DEFAULT" ItemGroupCode="PS" >
   <PrimaryInformation PricingQuantityStrategy="IQTY">
   <ComplexQuery Operator="OR">
      <And>
         <Or>
         <Exp Name="UnitOfMeasure" Value="EACH"/>
         <Exp Name="UnitOfMeasure" Value="HR"/>
         </Or>
         <And>
         <Exp Name="ManufacturerName" Value="YANTRA"/>
         </And>
      </And>
   </ComplexQuery>
   </PrimaryInformation>
</Item>
```

The `OrganizationCode` and `ItemGroupCode` are the two attributes of
the `<Item>` element and `PricingQuantityStrategy` is the attribute
of the `<PrimaryInformation>` element considered in this example.
However you can include any or all of the attributes in the

getItemList API. All the attributes in the API are interpreted with an implied And along with the complex query operator.

Apply the following rules when including complex queries:

- You can define only one ComplexQuery under a single element. For example, you cannot have two ComplexQuery operator under an Item element.

- You cannot add a single complex query against two different tables. For example, in getShipmentList API you cannot use ChainedFromOrderHeaderKey and ShipmentLineNo in the same query, since the former belongs to YFS_ORDER_LINE table and the latter is an attribute of the YFS_SHIPMENT_LINE table.

- The attribute with no value is not considered in the complex query, like Attribute="".

- There can be only one element under the ComplexQuery namely, And or Or.

- And or Or elements can have one or many child elements as required.

- And or Or elements can have other And or Or expression elements as child elements.

This example can be interpreted as the following logical expression:

```
(OrganizationCode="DEFAULT" AND ItemGroupCode="PS" ) AND
((PricingQuantityStrategy="IQTY") OR ( ( UnitOfMeasure =
"EACH" OR UnitOfMeasure="HR" ) AND ( ManufacturerName =
"YANTRA") ))
```

Thus by following the above example you can include complex queries to achieve desired results from your database using the above mentioned APIs.

8.3 Time Triggered Transactions

A time-triggered transaction, or agent, is a program that performs a variety of individual functions, automatically and at specific time intervals. It is not triggered by conditions, events, or user input.

There are three types of time-triggered transactions:

- Business process transactions - responsible for processing day-to-day transactions
- Monitors - watch and send alerts for processing delays and exceptions
- Purges - clear out data that may be discarded after having been processed

For information on using the time triggered transactions provided by Sterling Supply Chain Applications, see the *Sterling Supply Chain Applications Platform Configuration Guide*. For information on creating custom time-triggered transactions, see Section 8.2, "APIs" on page 459.

8.3.1 Extending Standard Transactions

You can extend the following the transactions provided by Sterling Supply Chain Applications by using one of the following mechanisms:

- User exits
- Event handlers

8.3.2 Using User Exits to Extend Standard Transactions

Sterling Supply Chain Applications provide ability to extend or override key business algorithms. This is accomplished through user exits that are invoked when the algorithms are executed.

A typical user exit can:

- Override application logic by providing its own logic.
- Extend application logic by providing more inputs to the application algorithm.

For example, if an order is split into multiple shipments, you may need to compute the order price differently for each shipment. In order to change the way Sterling Supply Chain Applications compute order prices, you can override the specific computation or algorithm in the `repriceOrder` user exit.

Each user exit is a separate Java interface. You may choose to implement only those user exits where you want to override or augment the business logic.

> **Note:** When the API or time-triggered transaction is executing the default algorithm, the user exit is called. If your implementation of the function throws an exception, the current transaction is rolled back.

For detailed descriptions of each user exit, see the following packages in *Sterling Supply Chain Applications Javadocs*:

- com.yantra.ycm.japi.ue
- com.yantra.ycp.japi.ue
- com.yantra.ydm.japi.ue
- com.yantra.yfs.japi.ue

8.3.2.1 Implementing and Deploying User Exits

All user exit classes should be deployed as a JAR file that is available in the CLASSPATH of the agent adapter script and in the `yantra.ear` file deployed in your application server.For more information about implementing user exits, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

For information on creating and deploying custom classes, see the *Sterling Supply Chain Applications Installation Guide*.

8.3.2.2 Guidelines for Usage of User Exits

The following guideline has to be kept in mind when you are using User Exits within Sterling Supply Chain Applications API:

- User Exits are structured to return specific information and hence their usage must be restricted for such purpose alone.

- From the user exits, you cannot invoke APIs that modify the data. This is to ensure that errors do not occur because of the data that is getting modified in the parent transaction (the transaction that calls the user exit), and the same data getting modified in the user exit custom code. For example, you cannot invoke a `changeOrder()` API from an user exit that obtains information related to the same order.

However, APIs that do not modify the data (like select APIs) can be invoked in the user exits. For example, you can call `getOrderDetails()` API from an user exit.

8.3.3 Using Event Handlers to Extend Standard Transactions

Sterling Supply Chain Applications raise events at specific moments in processing. Sterling Supply Chain Applications enable you to define an action to be performed when a specific event occurs.

In Sterling Supply Chain Applications configuration, an event is defined to invoke the Sterling Supply Chain Applications event handler. The event handler performs special processing on data published by the event before being transported to the transport services layer.

As part of the event configuration, services can be invoked. When such services are invoked, some of the events, like INVENTORY_CHANGE pass data as a map whereas the other events pass data as an XML. In the event of data being passed as a map, when the service is configured for such an event the data map is converted to an XML as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<YantraXML>
    <XML AccountNo="" AdjustmentType="ADJUSTMENT"
        EnterpriseCode="DEFAULT" ItemID="ITEM1"
        ItemKey="2005030116023851364" ...... />
</YantraXML>
```

8.3.4 Configuring Events

In Release 5.0 and later, you associate an action to a service to perform special processing required. For example, the event PUBLISH_SHIP_ADVICE could invoke the Sterling Supply Chain Applications event handler to call a custom Java class. The Java class can augment the publish ship advice XML. The Sterling Service Definition Framework can transport the enriched XML data to the transport services layer.

For more information on configuring actions and associating the actions to events, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

For more information on the various components you use to build a service, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

The event handlers that can be invoked from an action (configured in the Other tab) are described below.

> **Note:** The following event handlers are provided exclusively for backward compatibility purposes.

8.3.4.1 E-Mail Message Event Handler

Sterling Supply Chain Applications provide standard Send Email event handler for sending e-mail messages on various events. For example, when a line is shipped, you can send shipment information. SMTP-based e-mail is supported for these messages. When configured in the Other tab when configuring an Action, the e-mail is sent synchronously. If the mail server is not available, an exception is thrown. For sending an event notification in the form of an e-mail message using a service, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

8.3.4.2 E-Mail Templates

E-mail templates enable you format e-mail messages. The E-mail node receives XML data and merges it with the XSL template you specify. You can configure an E-mail node in the Service Builder to use an e-mail template.

To create and use a custom e-mail template:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/template/email/orders_ mail.xsl` template file.

2. Modify the file as needed, rename it, and save it within the same directory. You can save it to another directory, but using the standard directory structure supplied by Sterling Supply Chain Applications helps ensure consistency.

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4. From the Service Builder, configure an E-mail node to use your custom e-mail template. In the Body Template field specify either the file name or the path:

   — If you want to specify the ***path***, use the relative path from `<YFS_HOME>`. For example, if your template is in the `<YFS_HOME>/template/email/` directory, specify `template/email/<custom_email_template>.xsl`.

   — If you want to specify the ***file name***, ensure that your CLASSPATH contains the path to the template. For example, if you save the template to the `template/email/` directory, your CLASSPATH must contain `YFS_HOME/template/email/`.

Note that using the `<YFS_HOME>/Applications/Foundation/template/email/<file_name>.mlt` file with Actions to accomplish the same purpose has been deprecated as of Release 5.0. Use the E-mail node and an XSL file instead.

8.3.5 Alert Console Event Handler

Sterling Supply Chain Applications support standard event handlers sending exception alerts to the Alert Console. These exceptions are redirected to a specific user or to an exception queue for a group of users. For example, you can send an exception alert to the customer service representative queue when authorization fails so the customer service representative can contact the buyer in person. For more information about the Alert Console, see the *Sterling Supply Chain Applications Platform User Guide*.

8.3.6 Exception Alert Templates

Exception alert templates enable you to supply additional text to alerts raised. This enables you to make error message more descriptive and easy to understand. They also provide a means of supplying a hyperlink to the resolution screens from the Alert console. For example, for any alert created for an order, shipment, or load document type, a hyperlink is created and displays in the "Created For" column on the Alert List screens. In the Exception Alert Template you can customize this hyperlink or create any other hyperlinks. The input data to the alert node is merged with the template you specify and then posted as the description of the alert raised.

Events that publish data in data buffer format use an ECT template, which are text files that contain tags. These tags are replaced by actual data at run time. All tags use the |#YFS_<tagname>| syntax.

For example, use the tag |#YFS_OrderNo| to have the actual order number appear in its place (if OrderNo is published as a part of the data buffer). Any of the data elements published in the data buffer can be used in the template.

For the exception console, there are two templates. One template is used for determining the DETAIL_DESCRIPTION field of the YFS_INBOX table. The other template is used for determining the LIST_DESCRIPTION field of the YFS_INBOX table. The templates are merged with the data published (by an event) and the resulting string is populated to the corresponding field.

To create and use a custom exception template:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/template/exception_console/ example_exception_console.xsl` template file.

2. Modify the file as needed, rename it, and save it within the same directory. You can save it to another directory, but using the standard directory structure supplied by Sterling Supply Chain Applications helps ensure consistency.

3. Update the Sterling Supply Chain Applications runtime. For more information about updating Sterling Supply Chain Applications runtime, see the *Sterling Supply Chain Applications Installation Guide*.

4. From the Service Builder, configure an Alert node to use your custom exception console template. In the XSL Template field, specify either the file name or the path:

   – If you want to specify the *path*, use the relative path from <YFS_ HOME>. For example, if your template is in the `<YFS_ HOME>/template/exception_console/` directory, specify `template/exception_console/<custom_alert_template>.xsl`.

   – If you want to specify the *file name*, ensure that your CLASSPATH contains the path to the template. For example, if you save the template to the `template/exception_console/`

directory, include `<YFS_HOME>/template/exception_console/` in your CLASSPATH.

Note that using the `<YFS_HOME>/Applications/Foundation/template/exception_console/<file_name>.ect` file with Actions to accomplish the same purpose has been deprecated as of Release 5.0. Use the Alert node and an XSL file instead.

8.3.7 Publish Event Handler

Sterling Supply Chain Applications enable publishing of data by the event manager to external systems for interoperability. You can configure any event to publish data to other systems. For example, upon the event ON_SUCCESS of the SHIP_CONFIRM transaction, you may want to publish data to an external financial system.

When configuring the Publish event handler, you must provide a set of system IDs separated by semicolons (for example, `SYSTEM1;System2;TestSystem`). These are IDs of time-triggered Transactions that are expected to read and process the data. The event manager writes the data provided by the transaction to the YFS_EXPORT table and writes one record for each system ID configured.

For publishing data using a service, use a Database transport node. For more information, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

8.3.8 Execute Event Handler

The Execute event handler tells the event manager to execute the specified program. Sterling Supply Chain Applications look for this program in the PATH. Data is passed to the program as the first command line argument. Like the Publish event handler, the Execute event handler is executed asynchronously. This is achieved by using a custom API that executes the program in a service.

8.3.9 Java Extension

Java extensions enable you to implement event handlers as Java classes. With each action, you can associate one Java event handler. The Java class you create must implement the `com.yantra.yfs.japi.util.YFSEventHandlerEx` interface. This is

achieved by using a custom API which actually executes the program in a service.

8.3.10 Database Extension Using Stored Procedures

If you configure an event handler as an Oracle stored procedure, the stored procedure is invoked with the following parameters:

```
PROCEDURE DO_ACTION(TRANID IN VARCHAR2,ACTIONCODE IN VARCHAR2,KEY_DATA IN
VARCHAR2,DATA_TYPE IN NUMBER,DATA_BUFFER1 IN VARCHAR2,DATA_BUFFER2 IN
VARCHAR2,DATA_BUFFER3 IN VARCHAR2,DATA_BUFFER4 IN VARCHAR2,DATA_BUFFER5 IN
VARCHAR2,DATA_COMPLETE IN NUMBER,SHIP_NODE IN VARCHAR2,RETURN_VALUE IN OUT
NUMBER)
```

PL/SQL limits the maximum size of a VARCHAR2 variable to 2,000 bytes, so 2,000 bytes is the maximum length of the string passed in data buffers. If the input is greater than 10,000 characters, the buffer is truncated and the parameter DATA_COMPLETE has a value of 0 (zero). If the buffer is less than 10,000 bytes and is completely passed to the stored procedure, DATA_COMPLETE is passed as 1.

> **Important:** You must not do a commit or a rollback in the stored procedure at any time in a stored procedure associated to a event handler.

Execution of stored procedures is not a supported service component.

8.3.11 HTTP Extension

If you configure an event handler as an HTTP extension, Sterling Supply Chain Applications send data using the post() function to the specified URL. Data is posted using the variables in Table 8–8.

Table 8–8   Variables Associated with HTTP Extensions

Variable	Description
sTranID	ID of the transaction raising the event.

Table 8–8   Variables Associated with HTTP Extensions

Variable	Description
iDataType	If iDataType is set to 1, XML data is published with the event and sData contains the entire XML string. Otherwise, iData Type is set to 0 and sData contains a name=value pair. To see what the name=value pair contains, see the DBD file associated with the event in *Sterling Supply Chain Applications Javadocs*.
sData	See iDataType description.
sShipNode	Ship node ID, if available.

A service can be configured to post data using the HTTP protocol. For details, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

8.3.12 COM Extension

Using COM requires setting up your server and runtime clients as described in Using Sterling Supply Chain Applications with Microsoft COM+.

If you define an event handler as a COM extension, Sterling Supply Chain Applications call COM component you specify as the COM object. The COM component you create must expose the IDispatch interface and a pre-identified function doAction() as part of the interface. The signature of the doAction() function should be as follows:

```
[IDL Syntax]
doAction ( [in] BSTR sTranID, [in] BSTR sActionCode, [in] BSTR sKeyData,
[in]long DataType, [in] BSTR sData, [in] BSTR ShipNode, [Out] long *Retval);
```

Because Sterling Supply Chain Applications access this function through the IDispatch interface, you can write your COM component either in the Visual Basic or C++ programming language. The function signature that you must expose through Visual Basic is:

```
Public Function doAction(ByVal bsTranID As String, ByVal boActionCode As String,
ByVal bsKeyData As String, ByVal DataType As Long, ByVal bsData As String, ByVal
ShipNode As String, retVal as Long) As Long
```

The function signature that you must expose through C++ is:

```
STDMETHODIMP COMActionImpl::doAction(BSTR sTranID, BSTR sActionCode, BSTR
```

```
sKeyData, long DataType, BSTR sData, BSTR Node, long *RetVal)
```

`COMActionImpl` is the class name that implements the `doAction()` function. While configuring the event handler, you must specify the Program ID of the COM component to be invoked.

8.3.13 Event Chaining

Some event handlers support invocation of APIs. These APIs can retrieve more data pertinent to the event being raised, or they may direct to perform a business computation or transaction. In turn, the transaction can raise other events, thus resulting in event chaining. For example, one of the event handlers for the ORDER_CREATE event can check the validity of an order and call an API to HOLD the order. This results in the triggering of the HOLD event, which sends an e-mail message to a customer service representative. Event chaining provides an extensibility mechanism that is required for complex business processing.

> **Important:** Actions that are configured to be invoked by the ON_FAILURE event must not perform any database updates. Any database updates performed by an action invoked by the ON_FAILURE event is rolled back.

To access data published by Sterling Supply Chain Applications, you must implement the `YFSEventHandlerEx()` interface, which provides the `handleEvent()` function to implement as follows:

```
public boolean handleEvent (YFSEnvironment oEnv, String sTranID, String
  sActionCode, Map sKeyData, int iDataType, Object sData, String sShipNode,
  String [] parms) throws Exception
```

> **Important:** While it is possible to implement an API that causes an event that is associated with an action in Sterling Supply Chain Applications to call the same API, take care to avoid this situation. It creates an infinite loop.

The class name must be configured as the Java object on the Action Configuration screen in Sterling Supply Chain Applications transaction

configuration. For more information, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

All parameters for the `handleEvent()` function are input parameters with values inserted by Sterling Supply Chain Applications. The last parameter in this function is the `parms` parameter, which is an array of String constants. The values of these string constants are the values of the string constants specified after the class name during the action configuration in Sterling Supply Chain Applications.

> **Note:** For details on the *YFSEventHandlerEx* interface, see the *Sterling Supply Chain Applications Javadocs*.

8.4 Customizing Condition Builder Fields

The Sterling Supply Chain Applications condition builder is used as a part of the service definition framework or in pipeline definitions. This can be used if you want to define dynamic conditions but also use some static set of attributes present in the condition builder. The following section explains the use of the fields that can be customized in the condition builder.

The following features can be customized while creating conditions:

- Adding Custom Attributes by Process Type
- Adding Custom Attributes during Condition Definition
- Providing Condition Properties in Dynamic Condition
- Providing Condition Cases for an Advanced XML Condition
- Creating and Modifying Advanced XML Conditions

> **Note:** All the string comparisons made in the condition builder are case sensitive.

For more information on defining conditions and using condition builder for services, refer to *Sterling Supply Chain Applications Platform Configuration Guide*.

8.4.1 Adding Custom Attributes by Process Type

You can add custom attributes in the statement builder, based on the process types such as order fulfillment, outbound shipment and so forth. This customization is useful when you want to evaluate a condition based on an attribute that is not provided as a pre-defined set in the condition builder. These custom attributes are added in the condition builder when you create a condition. For more information on creating conditions refer to *Sterling Supply Chain Applications Platform Configuration Guide*.

The custom attributes can be added by creating a custom XML file as below:

1. Create a directory named `extn` in `<YANTRA_ HOME>/Applications/Foundation/template/configapi` directory.

2. Create a file named `extn_conditionbuilder.xml` in `<YANTRA_ HOME>/Applications/Foundation/template/configapi/extn` directory. The format of a sample XML file is given in Example 8–9.

> **Note:** For customizing condition builders for order hold types create a file called `holdtype_extn_ conditionbuilder.xml` and follow the same procedure outlined in this section.

> **Note:** You must restart your application server, every time you add conditions in the `extn_ conditionbuilder.xml` file, for the changes to appear in the configurator screen.

The sample file creates conditions for extended order number, shipping information and order details. The extended order details has further sub-elements such as, `CustomItemId` and `ExtnOrderType`.

The Id attribute must be identical to the element name. For example, the element `ExtnOrderNo` must contain the attribute `Id="ExtnOrderNo"`.

Example 8–9   Sample File for Adding Custom Attributes

```
<CustomConditionAttributes>
    <ProcessType Name="ORDER_FULFILLMENT">
        <ExtnOrderNo Id="ExtnOrderNo" DisplayName="Extended Order Number"
        DataType="Text-40" Type="Leaf"/>
        <ExtnShiptoId Id="ExtnShiptoId" DisplayName="Extended Ship To Id"
        DataType="Text-40" Type="Leaf"/>
        <CustomOrderDetails Id="CustomOrderDetails" DisplayName="Custom Order
        Details">
            <CustomItemId Id="CustomItemId" DisplayName="Custom Item Id"
            DataType="Text-100" Type="Leaf"/>
            <ExtnOrderType Id="ExtnOrderType" DisplayName="Extended Order Type"
            DataType="Text-100" Type="Leaf"/>
        </CustomOrderDetails>
    </ProcessType>
</CustomConditionAttributes>
```

Table 8–9   Attributes Details

Attribute	Description
ProcessType	
Name	Required. Specify the name of the Process type where you are creating the condition.
Elements	
DisplayName	Required. Specify the display name for this attribute in the condition builder screen.
DataType	Required. Specify the data type needed for this attribute. For example, the pre-defined attribute BillToId has a data type of "ID-40".
	The data type can be any one of the types provided in the `<YANTRA_ HOME>/Applications/Foundation/template/api/DataTypes.xml` file.

Table 8–9   Attributes Details

Attribute	Description
Id	Required. Specify the ID you want this condition to be associated. By default this is same as the element name.
Type	Required. Specify the type of this attribute. For example "Leaf" type identifies the attribute as part of a menu item.
	If it is not specified, the attribute is considered to be a menu header leading to a sub menu. In Example 8–9 the element CustomOrderDetails does not have a type defined, since it has leading sub-menus such as CustomItemId and ExtnOrderType.

3. The elements like ExtnOrderNo, ExtnShipToId, etc., as shown in Example 8–9, are used to define the conditions when it is saved in the database. Therefore, the element name must be unique for a single process type.

4. This XML file content is merged with the pre-defined static attributes by process type and displayed on the screen as shown below:

5. However, upon clicking OK in the statement builder window, the display name of the XML is shown in the condition detail window.

6. The display names are replaced with their element names once the condition is saved. These element names along with their conditions

is referred as the condition value in the database table YFS_
CONIDITION.

7. These custom attributes is shown in a similar way as the current
   static attributes in condition builder.

   The custom attributes defined in this manner can be used as part of
   the service definition framework or in pipeline definitions for creating
   events or conditions.

Note: However, custom attributes cannot be used to
evaluate conditions in pipeline determination rules.

8.4.2 Adding Custom Attributes during Condition Definition

When you define conditions for a particular order, you can optionally
choose to add your own custom attributes to be evaluated as part of the
condition.

1. The custom attributes can be added in the condition builder by
   clicking Choose Field and selecting Enter Your Own Attribute
   option in the statement builder window.

2. Enter the attribute name you want to be included in the condition as
   shown below, and click Add.
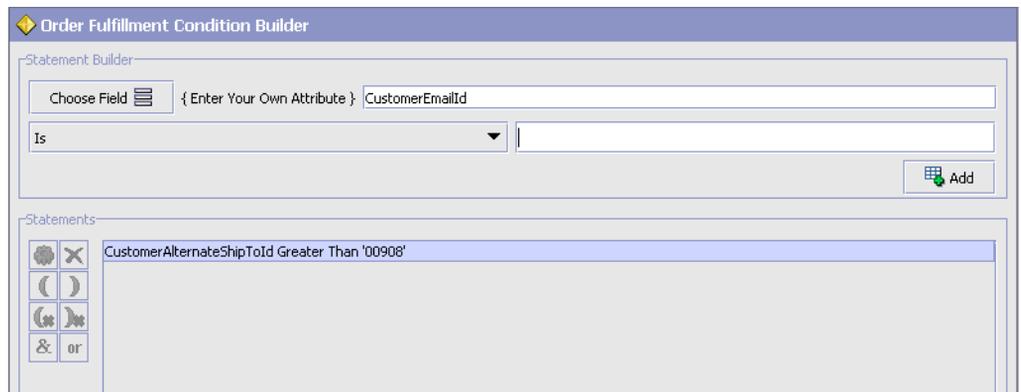
For example, if you want to include a attribute named `CustomerEmailId`, enter that attribute in the text box provided and continue with your statement creation.This attribute is included in evaluating the condition.

> **Note:** This field is limited only to unexposed key attributes that are pre-defined by Sterling Supply Chain Applications as opposed to any XML attribute that you can enter.

3. Once the name is entered, the rest of the condition can be built in the same way as the pre-defined attributes. For more information on how to create a condition using pre-defined attributes, see *Sterling Supply Chain Applications Platform Configuration Guide*.

   The custom attributes created during condition definition can be used as part of the service definition framework or in pipeline definitions for creating events or conditions.

   > **Note:** Custom attributes defined in this manner is available only when defining this condition. The attribute entered is not available for re-use in other conditions.

8.4.3 Providing Condition Properties in Dynamic Condition

The condition definition enables you to create static or dynamic or advanced XML condition.

When creating dynamic conditions you must check Dynamic field and mention the class name to be executed to evaluate the condition. This creation of dynamic condition is extended to include configuration of custom name, value properties. These properties are set into the java class before evaluating the condition.

To enable these attributes the dynamic condition class should be implemented a YCPDynamicConditionEx interface. The definition of this interface is as follows:

```
public interface YCPDnamicConditionEx {
    boolean evaluateCondition(YFSEnvironment env, String name, Map mapData,
    Document doc);
    void setProperties(Map map);
}
```

The parameter name passed to the interface refers to the Condition Name configured during definition.

For more information about the interface and parameters, see the *Sterling Supply Chain Applications Javadocs.*

8.4.4 Providing Condition Cases for an Advanced XML Condition

When you are modifying decision table-based advanced XML conditions, you can extend an advanced XML condition to include different attributes of custom cases. These attributes are set before you evaluate the decision table-based advanced XML condition of type.

You can add or modify existing cases defined for a decision table-based advanced XML condition. You can modify the different attributes of a condition case. After you modify a condition case and click the Save button, the new values of the attribute reflects in both Advanced XML and Source View screen.

The default return value defined for a decision table-based advanced XML condition displays as a hyperlink on the Advanced XML screen. If none of the condition cases are satisfied, the Greex engine returns the default value. Click the hyperlink to edit the default return value and specify the new default return value for the advanced XML condition. The pop-up screen displays the old value. You can also enter new values. The new value reflects in the Advanced XML screen as well as in the Source View screen when you click the Save button in the pop-up screen.

8.4.5 Creating and Modifying Advanced XML Conditions

This section explains how to modify an advanced XML condition, which is also known as the Greex Rule. You can only assign new values to the modifiable parameters of an advanced XML condition. You can also localize or log information about an advanced XML condition or Greex Rule.

8.4.5.1 What is Greex?

The Greex framework enables you to define advanced XML conditions based on the Greex syntax. The Greex framework provides a declarative mechanism to use an input XML document in order to evaluate advanced XML conditions.

Some of the salient features of the Greex framework are:

*   It is XML aware.

*   It supports namespaces.

*   It contains a set of libraries.

*   It can return an XML element, a String, or Boolean value.

The Greex framework contains the Greex library, which is comprised of a set of functions you can call on input data through XPath expressions. Constants can be used in function parameters also.

8.4.5.2 What is Greex Syntax?

Using the Greex syntax you can create advanced XML conditions or Greex Rules. An advanced XML condition is used to evaluate certain conditions on the input data. The Greex syntax is an XML based. This style of condition evaluation enables you to use the input data in multiple ways, rather than just a Boolean output. The Greex syntax provides Greex constructs that are capable of being nested using multiple IF, and ELSE blocks, and also allows to group expressions using an AND or OR operator. Each expression comprises one or more function calls. You can include functions in a nested loop, which means parameters to functions can be other function calls. These contain basic IF ELSE conditions. Table 8–10 describes various elements of an If/Else construct.

Table 8–10   If/Else Condition's Element List

Element	Description
Condition	The Condition element provides a logical grouping of all expressions that need to be evaluated by operating on an input XML.

Return	Every condition defined under the If/Else construct must return a value. A condition can return an XML element, string, or boolean value. It returns the appropriate value for an If/Else condition in the Return element. For example,
	
	 • If you want to return an XML element, the Return element can be:
	
	 `<Return>`
	
	 `<Value output="<Order`
	
	 `type="Web"`
	
	 `discount="5"/>"/>`
	
	 `</Return>`
	
	 • If you want to return a string, the Return element can be:
	
	 `<Return>`
	
	 `<Value output="Draft Order`
	
	 `Created">`
	
	 `</Return>`
	
	 • If you want to return a boolean value, the Return element can be:
	
	 `<Return>`
	
	 `<Value output="true">`
	
	 `</Return>`

The Condition element contains various Expression elements. Each Expression element defines the expression that you want to evaluate for

the specified condition. Table 8–11 describes various elements of a condition.

Table 8–11   Condition's Element List

Element	Description
Expression	The Expression element contains the expressions that you want to evaluate for a condition to satisfy. To make function calls in the expression, prefix the function name with "fn:". For example, define the following Expression element which makes a function call:
	`<Expression>fn:equals(@ZipCode,"01876")</Expression>`
	You can also pass functions as parameters to other functions. For example, you can define the following Expression element to pass functions as parameters to other functions:
	`<Expression>fn:intGreater(fn:count(OrderLines/OrderLine), "100")</Expression>`
Group	This is an optional element. If you want to evaluate a set of expressions together, you must group these set of expressions together.
	The Group element's "op" attribute indicates the operation you want to perform on the set of expressions. The valid values are: "or" and "and". You can define more than one expression in a Group element. If the Group element's op value is "and", then the condition satisfies only if ALL expressions that are part of the Group element evaluates to "true". Likewise, if the Group element's "op" attribute is "or", then the condition satisfies only if ONE of the expressions that are part of the Group element evaluates to "true".
	You can create any level of nested Group and Expression elements.
	Note: If you want to evaluate a single expression, define a single Expression element under the Condition element, without creating the Group element.

A sample IF ELSE construct for defining an advanced XML condition or Greex Rule can be:

```
<If>
 <Condition name="isWebOrder?">
  <Group op="and">
    <Expression>fn:!equals(@orderType,"WEB")</Expression>
    <Expression>fn:equals(address::@ZipCode, "01876")</Expression>
   </Group>
 </Condition>
 <Return>
        <Value output="&lt;Order type=&quot;Web&quot;
              discount=&quot;5&quot;/&gt;"/>
 </Return>
</If>
<Else>
 <Return>
        <Value output="&lt;Order type=&quot;Catalog&quot;
              discount=&quot;2&quot;/&gt;"/>
 </Return>
</Else>
```

8.4.5.3 Localizing the Greex Syntax

You can localize the Greex syntax as needed. For more information about localizing the Greex syntax, see the *Sterling Supply Chain Applications Localization Guide*.

8.4.5.4 Logging Information

You can log information about a Greex condition or an advanced XML condition at different levels. The different levels for logging information is described in the GreexLogConstants. You can log information ranging from Apache's log4j framework to a simple System.out.println().

To log information:

1. Create the GreexLogger class and implement the following methods within the class:

```
log(GreexLogData data) method and log information as needed.
For example:
public class MyLogger implements GreexLogger
{
     public void log(GreexLogData data)
     {
      System.out.println("Message:: "+data.getMessage());
         System.out.println("Severity:: "+data.getSeverity());
     }
```

```
   }
```

2. Register the GreexLogger class with the GreexContext using the registerLogger() method. For example,

```
public class MyApp
{
 GreexContext  ctx = new GreexContext();
 Ctx.registerLogger(new MyLogger(),GreexLogConstants.GREEX_DEBUG);
}
```

8.4.5.5 Creating an Advanced XML Condition using the Sterling Greex Editor

Using the Greex editor you can conveniently create and modify an advanced XML file, which is also known as the `Greex` file. The advanced XML file is used to define new advanced XML conditions to evaluate them on the input data.

Creating an advanced XML condition using the Sterling Greex Editor involves:

- Installing Prerequisite Software Components

- Creating a Java Project

- Running the Greex Model Wizard

- Creating a New Advanced XML Condition

- Validating an Advanced XML Condition

- Loading Advanced XML Conditions into the Database

- Importing Advanced XML Conditions from the Database

8.4.5.5.1 Installing Prerequisite Software Components

This section describes the various software components required for creating an advanced XML condition or a Greex rule using the Sterling Greex Editor. Before creating an advanced XML condition using Sterling Greex Editor, ensure that you have already installed the following software components:

- **Eclipse SDK**

  Install the Eclipse SDK version that Sterling Commerce supports. For more information about the Eclipse SDK version, see the *Sterling Supply Chain Applications Installation Guide*.

- **Eclipse Related Plug**

  Install the Eclipse Modelling Framework (EMF) plug-in version that Sterling Commerce supports. For more information about the Eclipse EMF plug-in versions, see the *Sterling Supply Chain Applications Installation Guide*.

- **Java Development Kit (JDK)**

  Install the JDK version that Sterling Commerce supports. For more information about the JDK version, see the *Sterling Supply Chain Applications Installation Guide*.

  > **Note:** If you are using a browser with JRE version lower than 1.5, the exception java.security.AccessControlException displays when you try to open the greex rule using Sterling Supply Chain Applications Configurator.
  >
  > To resolve this error, in the `<JRE_ HOME>/lib/security/java.policy` file, add the following property under the default permission for all domains section:
  >
  > `permission java.util.PropertyPermission "java.home", "read";`
  >
  > where `<JRE_HOME>` is the JRE installation directory.
  >
  > Now, close all browser instances and again log in to Sterling Supply Chain Applications.

- **Sterling Greex Plug**

  Install the Sterling Other Tools plug-in that Sterling Commerce supports. For more information about the Sterling Greex plug-ins version, see the *Sterling Supply Chain Applications Installation Guide*.

  This plug-in is shipped along with Sterling Supply Chain Applications that can be found at:

  `<YANTRA_HOME>/Applications/Foundation/othertools`

  where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.

> **Note:** If you are installing a new version of the Sterling Other Tools plug-ins or updating the previous versions you must clean the cached build information in Eclipse.
>
> To clean this information, start Eclipse with the "-clean" option:
>
> 1. Right-click on the Eclipse shortcut and select Properties from the pop-menu. The Properties window displays.
>
> 2. In Target, enter the command line argument "-clean" at the end. For example, `"C:\Eclipse 3.2\eclipse\eclipse.exe" -clean`.
>
> 3. Start Eclipse.

Installing the Sterling Other Tools plug-in

To install the Sterling Other Tools plug-in:

Extract the `<YANTRA_ HOME>/Applications/Foundation/othertools/com.yantra.yfc.other. tools.zip` file to the `<ECLIPSE_HOME>/plugins` folder.

where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications. <ECLIPSE_HOME> refers to the Eclipse SDK installation directory. Restart Eclipse SDK in order to allow the newly installed plug-ins to be found.

8.4.5.5.2 Creating a Java Project

Before you start working with the Greex editor, you must create a java project, which acts as a container for the Greex Model wizard.

To create a java project:

1. Start Eclipse SDK.

2. From the menu bar, select File > New > Project…. The New Project window displays.

3. From the list of wizards, select the Java Project.

4. Click Next. The New Java Project window displays.

5. In Project name, enter the name of the new java project.

6. Click Next. The Java Settings page displays.

7. Click Finish. The new java project is created.

8.4.5.5.3   Running the Greex Model Wizard

After creating the new java project, run the Greex Model wizard on top of the new java project that you created. The Greex Model wizard either creates an empty Greex file or advanced XML file with *.greex extension. You can create an advanced XML condition or Greex Rule in the Greex file.

To run the Greex Model wizard:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The java project displays in the Navigator view.

3. Expand the java project that you created. For more information about creating a java project, see Section 8.4.5.5.2, "Creating a Java Project".

4. Right-click on the folder where you want to store the Greex Rule XML file and select New > Other from the pop-up menu. The New window displays.

5. From the list of wizards, select Sterling RCP Wizards > Sterling RCP Greex Model.

6. Click Next. The New window displays.
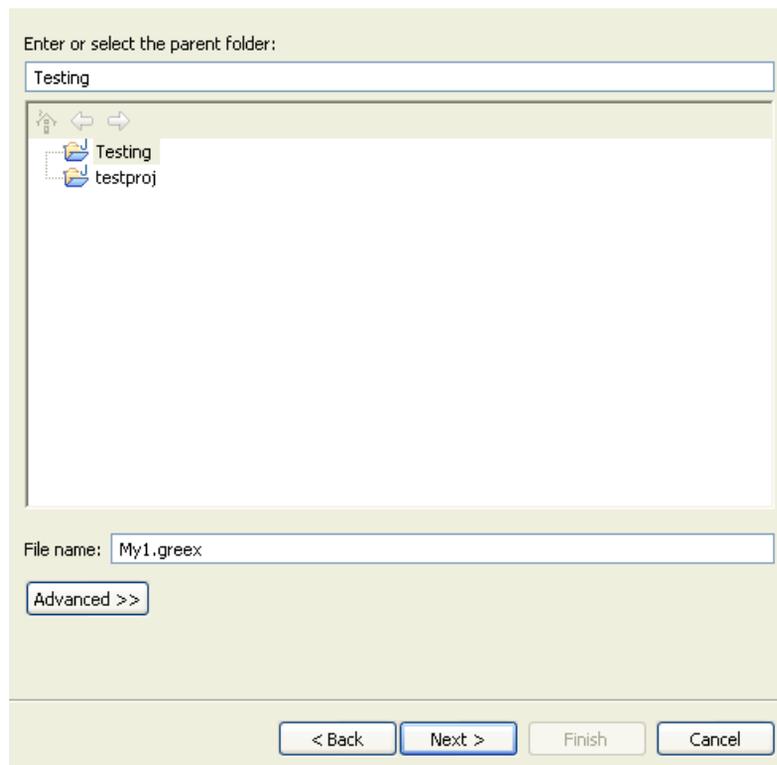
7. In File name, enter the name of the Greex Rule XML file. Make sure that the file name that you specify ends with ".greex" extension.

8. Click Next. The Greex Model window displays.
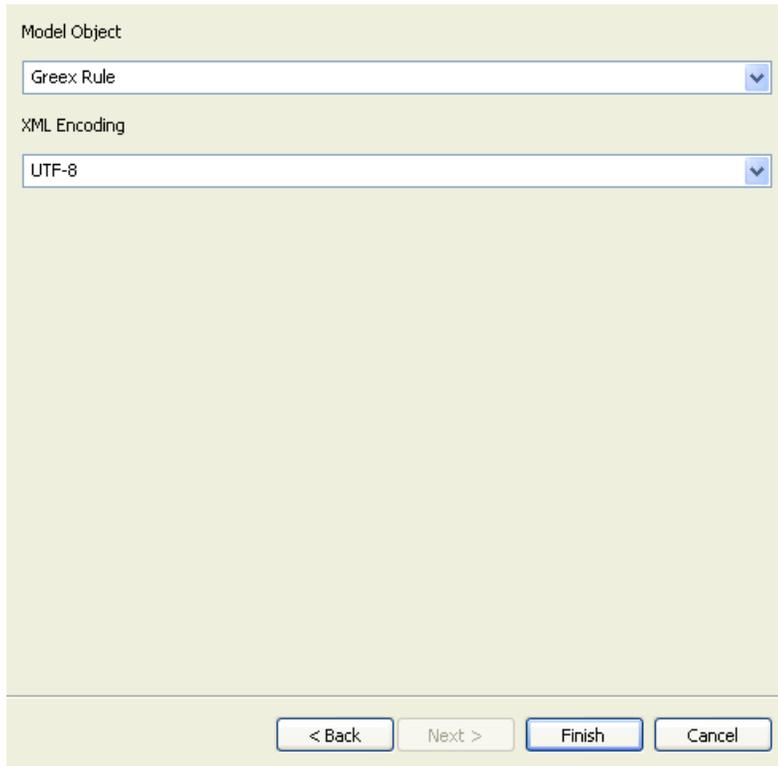
9. In Model Object, select Greex Rule.

10. In XML Encoding, select UTF-8.

11. Click Finish. The Greex Rule XML file is created in the specified folder.

8.4.5.5.4 Creating a New Advanced XML Condition

After creating the Greex file or an advanced XML file, you can define an advanced XML condition or Greex Rule. You can create two types of advanced XML conditions:

- Normal advanced XML condition
- Decision Table based advanced XML condition.

> **Note:** Make sure that you **do not** add any comments in the advanced XML condition or Greex rule of any type.

Creating a Normal Advanced XML Condition

The normal advanced XML condition is useful in scenarios where you need to define multiple condition criteria for an advanced XML condition you need to define multiple condition criteria and each condition criteria is associated with different attributes. These multiple condition criteria are defined using nested IF and ELSE constructs. For example, you may want to create a normal advanced XML condition for the condition criteria such as:

If Ordertype="WEB" and OrderQty="100", then TaxExemptFlag="N".

Else

    If OriginalTotalAmout="1000" and OrderLine>"5", then

      discount="10" TaxExemptFlag="Y".

    Else TaxExemptFlag="N".

In this case, we have multiple condition criteria and each condition criteria has different attributes associated with it. The first condition criteria has Ordertype, OrderQty, and Discount associated with it. Whereas, the second condition criteria has OriginalTotalAmount, OrderLine, and TaxExemptFlag associated with it.

These condition criteria are defined in the advanced XML file or Greex file using nested IF AND ELSE constructs. A sample advanced XML file or Greex file for the advanced XML condition is as follows:

```
<GreexRule desc="Determine Discount and Tax Exemption based on
                 various attributes"
           id="getDiscountTaxExempt"
           name="Get Discount and Tax Exempt"
           returnType="Xml"
           type="">
 <If>
  <Condition name="isDiscount20?">
   <Group op="and">
    <Expression>fn:equals(@orderType, &quot;WEB&quot;)</Expression>
    <Expression>fn:equals(@orderQty, &quot;100&quot;)</Expression>
   </Group>
  </Condition>
```

```
   <Return>
    <Value output="&lt;Order TaxExemptFlag=&quot;N&quot;/>"/>
   </Return>
</If>
 <Else>
    <If>
     <Condition name="TaxExempted?">
      <Group op="and">
       <Expression>fn:equals(@OriginalTotalAmout, &quot;1000&quot;)</Expression>
       <Expression>fn:intGreater(@orderLine,&quot;5&quot;)</Expression>
      </Group>
     </Condition>
     <Return>
       <Value output="&lt;Order discount=&quot;10&quot;
              TaxExemptFlag=&quot;Y&quot;/>"/>
     </Return>
    </If>
    <Else>
     <Return>
      <Value output="&lt;Order discount=&quot;20&quot;
              TaxExemptFlag=&quot;N&quot;/>"/>
     </Return>
    </Else>
  </Else>
</GreexRule>
```

To create a normal advanced XML condition:

1. Expand the java project that you created.

2. In the Project Explorer hierarchy, select the `*.greex` file. Right-click and select Open With > Greex Model Editor from the pop-up menu.

3. Expand the tree structure that appears in the Greex editor. Click on a node from the tree. All child leaves of the node are listed. The Document Root element contains the Greex Rule root element.

4. Select the Greex Rule root element in the Properties view. Enter the values for various attributes. See Table 8–14 for the description of various attributes of the Greex Rule root element.

5. In the Properties view, you can view various properties of the selected element.

   To open the Properties View:

From the menu bar, select Window > Show View > Other…. From the list of views under Basic, select Properties.

Table 8–12   Greex Rule Element's Attribute List

Attribute	Description
Desc	Enter the description of the Greex rule.
Id	Enter a unique identifier of the Greex rule.
Name	Enter the name of the Greex rule.
Return Type	Enter the return type of the Greex rule. For a normal Greex rule, the valid values are "Xml", "String", and "Boolean". The normal Greex rule can return an XML document, string, or boolean value.
Type	By default, a Greex rule is a normal Greex rule.

6. Under the Greex Rule root element, create a new IF ELSE construct element, as needed. Right-click on the Greex Rule root element and select New Child > If/Else from the pop-up menu. You can create any level of nesting of IF and ELSE constructs.

7. Select the If/Else element. In the Properties view, enter the name of the If element in the Name property.

8. Under the If/Else element, create a new Condition child element. Right-click on the If/Else element and select New Child > Condition from the pop-up menu.

9. Select the Condition element. In the Properties view, enter the name of the Condition element in the Name property.

10. As every Greex condition must return a value, under the If/Else element, create a new Return element and specify the appropriate return value for the associated condition. Right-click on the If/Else element and select New Child > Return from the pop-up menu.

11. Right-click on the Return element and select New Child > Value from the pop-up menu. The Enter the value pop-up window displays.

12. Enter the value that you want to return if the IF condition satisfies.

13. Select the Return element. In the Properties view, in the Default property, specify the default value (if necessary) that you want to

return if the IF condition is not satisfied. In the Output property, specify the value that you want to return if the IF condition satisfies.

14. Under the Condition element, create a new Expression element to specify the expressions that you want to evaluate for the condition to satisfy. Right-click on the Condition element and select New Child > Expression from the pop-up menu. The Edit Expression pop-up window displays.

15. In Expression, enter the expression you want to evaluate. You can make function calls in the expression by prefixing the function name with "fn:". You can also pass functions to other functions.

> **Note:** Press Ctrl+Space and select the expression from the drop-down list.

16. (Optional) If you want to evaluate a set of expressions together, you must group them. Under the Condition element, create a new Group element to group a set of expressions together. Right-click on the Condition element and select New Child > Group from the pop-up menu.

Now, you can add more than one expression to this group. Right-click on the Group element and select New Child > Expression from the pop-up menu.

You can also add a new Group element to an existing Group element. Right-click on the Group element and select New Child > Group from the pop-up menu.

> **Note:** You can create any level of nested Group and Expression elements.

17. (Optional) Select the Group element. In the Properties view, specify the operation that you want to perform on the set of expressions in the Op property. The valid values are: "or" and "and".

If you specify the Op property as "or", a condition is satisfied if any of the expressions specified in the group is "true".

If you specify the Op property as "and", a condition is satisfied only if all expressions specified in the group is "true".

18. Click Save.

Creating a Decision Table Based Advanced XML Condition

Decision table based advanced XML conditions are useful in scenarios where you have multiple nested condition criteria to be defined for an advanced XML condition but each condition criteria has same attributes associated with it. In such cases, you can write just one condition and have a table of parameters that works like a switch statement. In the decision table based advanced XML condition you can define multiple nested condition criteria in a single IF construct. Hence, there is no ELSE construct in case of decision table based advanced XML conditions. The IF construct will contain an array of constant values instead of one constant value as used to be in case of normal advanced XML conditions. The IF construct will have a table of parameters that works like a switch statement.

For example, you may want to create a decision table based advanced XML condition for the condition criteria such as:

If Ordertype="WEB" and OrderLineQty="200", then Discount="5".

Else

  If Ordertype="STORE" and OrderLineQty="500", then Discount="7"

  Else

   If Ordertype="CALL" and OrderLineQty="250", then Discount="3".

  Else

    default="0"

Table 8–13 describes the above scenario in the form of a decision table.

Table 8–13   Decision Table

Order Type	Order Line Quantity	Discount
WEB	200	5
STORE	500	7
CALLCENTER	250	3
default	0	

In this case, we have multiple condition criteria but each condition criteria has the same attributes associated with it. All the condition cases have Ordertype, OrderLineQty, and Discount attributes associated with it. Therefore, this advanced XML condition will have only one IF construct and it will contain an array of constant values.

> **Note:** For a decision table based Greex rule, it is mandatory to give a default return value. This value is returned if no IF condition gets satisfied.

These condition criteria are defined in the advanced XML file or Greex file using IF constructs. A sample advanced XML file or Greex file for the decision table-based advanced XML condition is as follows:

```
<GreexRule desc="Determine Discount based on some attributes"
           id="getDiscount"
           name="Get Discount"
           returnType="String"
           type="DecisionTable">>
 <If>
  <Condition name="isDiscount5?">
   <Group op="and">
    <Expression>fn:equals(@orderType,
               &quot;WEB|STORE|CALLCENTER&quot;)</Expression>
    <Expression>fn:equals(@orderLineQty, &quot;200|500|250&quot;)</Expression>
   </Group>
  </Condition>
  <Return>
   <Value default="0" output="5|7|3"/>
  </Return>
 </If>
</GreexRule>
```

To create a new decision table based advanced XML condition:

1. Expand the java project that you created.

2. In the Project Explorer hierarchy, select the *.greex file. Right click > Open With > Greex Model Editor.

3. Expand the tree structure that appears in the Greex editor panel. Click on a node from the tree, all the child leaves of the node are

listed. The `Document Root` element contains the root element `Greex Rule`.

4. Select the root element `Greex Rule` and in the Properties view, enter the values for various attributes. Table 8–14 describes various attributes of the root element `Greex Rule`.

    In the Properties view, you can view various properties of the selected element. To open the Properties View, from the menu bar, select Window > Show View > Other…. From the list of views under Basic, select Properties.

Table 8–14   Greex Rule Element's Attribute List

Attribute	Description
Desc	Enter the description of the Greex rule.
Id	Enter the unique identifier of the Greex rule
Name	Enter the name of the Greex rule.
Return Type	Enter the return type of the Greex rule. For a decision table based Greex rule, the only valid value is "String". The decision table based Greex rule can only return a String.
Type	Enter the Greex rule type. By default, it is a normal Greex rule. To make it a decision table based Greex rule, select "DecisionTable" from the drop-down list.

5. Under the root element Greex Rule, create a new IF construct element as per the requirement. Right-click on the root element `Greex Rule` and select New Child > If from the pop-up menu.

> **Note:** A decision table based Greex rule cannot have an ELSE construct. It can only have a single IF construct.

6. Select the `If` element and in the Properties view, enter the name of the `If` element in the `Name` property.

7. Under the `If` element, create a new `Condition` child element. Right-click on the `If` element and select New Child > Condition from the pop-up menu.

8. Select the `Condition` element and in the Properties view, enter the name of the `Condition` element in the `Name` property.

9. As every Greex condition must return a value, under the `If` element, create a new `Return` element and specify the appropriate return value for the associated condition in the `Value` element. Right-click on the `If` element and select New Child > Return from the pop-up menu.

10. Right-click on the `Return` element and select New Child > Value from the pop-up menu. The Enter the value pop-up window displays.

11. Enter the value which you want to return if the IF condition gets satisfied.

12. Select the `Return` element. In the Properties view, in the `Default` property, specify the default value which you want to return if the none of the IF condition does not gets satisfied and in the `Output` property, specify the value which you want to return if the IF condition gets satisfied.

> **Note:** For a decision table based Greex rule, it is mandatory to give a default return value.

For a decision table based Greex rule, you can define an array of constant values in the `Value` element. The multiple values are separated using the "|" operator. For example, 5|7|3.

13. Under the `Condition` element, create a new `Expression` element to specify the expression(s) that you want to evaluate for the condition to be satisfied. Right-click on the `Condition` element and select New Child > Expression from the pop-up menu. The Edit Expression pop-up window displays.

14. In Expression, enter the expression that you want to evaluate. You can make function calls in the expression by prefixing the function name with "fn:". You can also pass functions to other functions.

> **Note:** You can press Ctrl+Space and select the expression from the drop-down list.

For a decision table based Greex rule, you can define an array of constant values for a particular XML attribute in the `Expression` element. The multiple values are separated using the "|" operator. For example, fn:equals(@orderType, "WEB|STORE|CALLCENTER").

15. (Optional) Only if you want to evaluate a set of expressions together, you must group these set of expressions. Under the `Condition` element, create a new `Group` element to group a set of expressions together. Right-click on the `Condition` element and select New Child > Group from the pop-up menu.

Now, you can add more than one expressions to this group. Right-click on the `Group` element and select New Child > Expression from the pop-up menu.

You can also add a new `Group` element to a `Group` element. Right-click on the `Group` element and select New Child > Group from the pop-up menu.

> **Note:** Any level of nesting of Group and Expression elements is allowed.

16. (Optional) Select the `Group` element and in the Properties view, specify the operation that you want to perform on the set of expressions in the "op" property. The valid values are: "or" and "and".

If you specify the `Op` property as "or" then a condition gets satisfied if any of the expressions specified in the group evaluates to true.

If you specify the `Op` property as "and" then a condition gets satisfied only if all of the expressions specified in the group evaluates to true.

17. Click Save.

8.4.5.5.5 Validating an Advanced XML Condition

After creating an advanced XML condition, you must validate its syntax and structure. For example, if an advanced XML condition is of decision table type, it must contain only one IF construct and no ELSE constructs. Therefore, you must validate an advanced XML condition before loading it into the database.

To validate an advanced XML condition:

1. Expand the java project that you created.

2. In the Project Explorer hierarchy, select the `*.greex` file. Right-click and select Open With > Greex Model Editor from the pop-up menu.

3. Expand the Greex rule that appears in the Greex editor panel. Click on a node from the tree. All child leaves of the node are listed. The Document Root element contains the Greex Rule root element.

4. Select the Greex Rule root element and right-click. Select Validate Greex Rule from the pop-up menu.

If the Greex rule that you created uses correct syntax and structure, the message "Greex rule validation succeeded" displays. Otherwise, the system displays an appropriate error message.

8.4.5.5.6 Loading Advanced XML Conditions into the Database

After creating an advanced XML condition or Greex rule using Sterling RCP Greex Editor, you must load the Greex file. This file contains the newly created advanced XML condition in the database, so that you can edit the modifiable parameters of the advanced XML condition (if needed) using the Sterling Supply Chain Applications Configurator.

> **Note:** Before loading an advanced XML condition or Greex rule into the database, you must validate the advanced XML condition for a valid structure or syntax. For more information about validating an advanced XML condition, see Section 8.4.5.5.5, "Validating an Advanced XML Condition".

To load an advanced XML condition into the database:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The java project displays in the Navigator view.

3. Expand the java project that you created. For more information about creating a java project, see Section 8.4.5.5.2, "Creating a Java Project".

4. Right-click the folder which contains all `*.greex` files, which contains an advanced XML condition that you want to load into the database. Select Greex > Export Greex Rule to Database from the pop-up menu. The Export Greex Rule to Database window displays.

| Sterling Supply Chain Applications Runtime | C:\SupplyChainApps\Runtime | ... |

Use Connection

< Back   Next >   Finish   Cancel
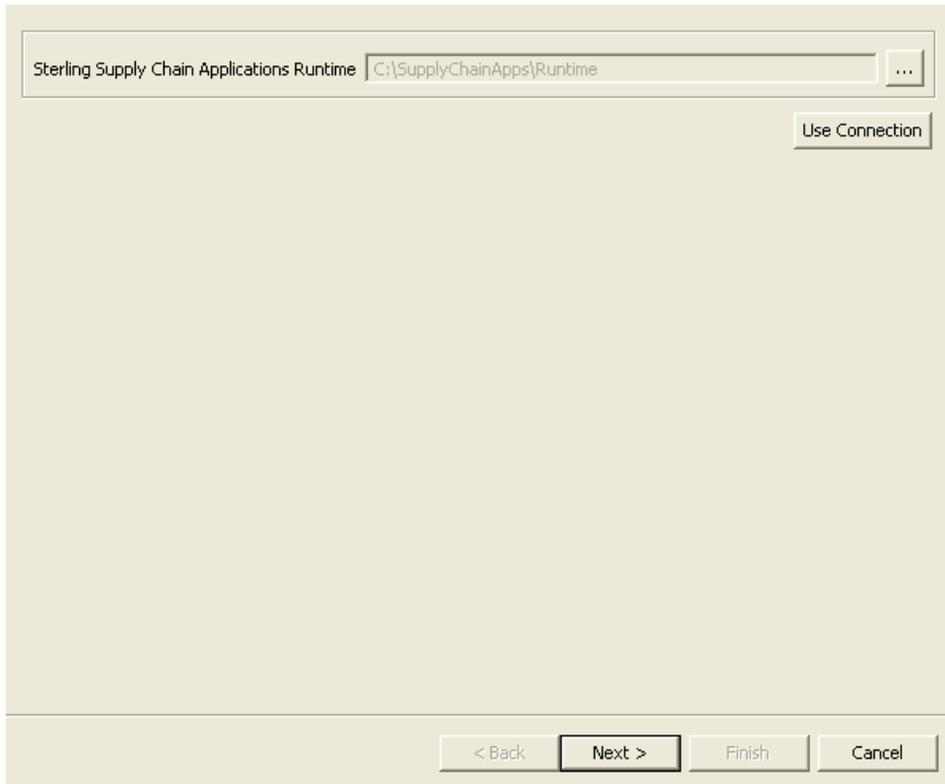
Table 8–15   Export Greex Rule to Database Window

Field	Description
Sterling Supply Chain Applications Runtime	Click and browse to the `<YANTRA_HOME>/Runtime` directory.
	where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications.
	The Greex rules is committed to the database specified in the `<YFS_ HOME>/Runtime/resources/yfs.properties` file.
	where <YFS_HOME> refers to the <YANTRA_ HOME>/Runtime directory.
Use Connection	Click this button to use the JDBC connection specified in the `<YFS_HOME>/Runtime/resources/yfs.properties` file.
	where <YFS_HOME> refers to the <YANTRA_ HOME>/Runtime directory.

5. Click Next. The Select the Process Type and Owner Key window displays.

| Process Type Key | 20030708142438541 ▼ |
| Owner Key | AIRB ▼ |

< Back   Next >   Finish   Cancel

Table 8–16   Select the Process Type and Owner Key Window

Field	Description
Process Type Key	Select the key of the appropriate process type from the drop-down list.
Owner Key	Select the key of the appropriate owner from the drop-down list.

6. Click Next. The Select Greex Rules window displays.

Table 8–17   Select Greex Rules Window

Field	Description
Process Type Key and Owner Key	Displays the process type and owner keys that you selected in the previous screen.
Select All	Click this button if you want to select all Greex files listed in List of Greex Files panel.
Unselect All	Click this button if you want to unselect all Greex files listed in the List of Greex Files panel.
List of Greex files	
Select the Greex files that you want to load into the database.	
Connection Id	Displays the unique identifier of a Greex rule defined in each Greex file that you select.

7. Click Next to create factory settings for the Greex rules defined in the each Greex file that you selected. The Select Factory Settings File window displays.
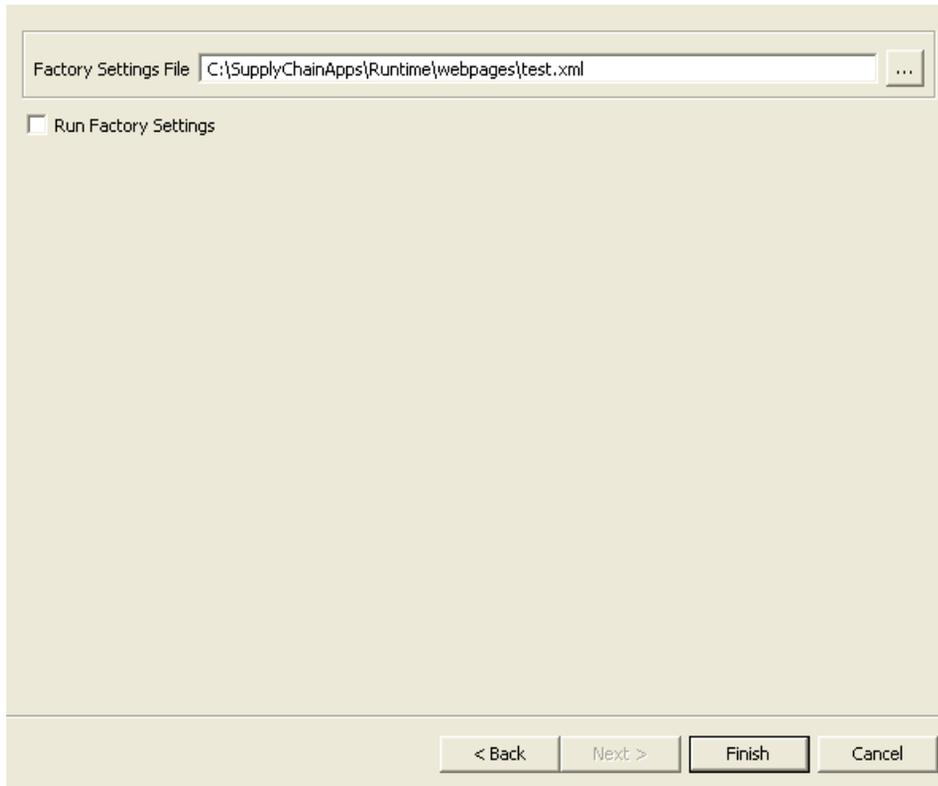
Table 8–18   Select Factory Settings File Window

Field	Description
Factory Settings File	Enter the path and name of the new factory settings XML file in which you want to store the YFS_CONDITION Factory Settings. Click ⎕. The Select XML File to Store the YFS_CONDITION Factory Settings pop-up window displays. Select the factory settings XML file in which you want to store the YFS_CONDITION Factory Settings.
Run Factory Settings	Check this box if you want to run the factory settings for the factory settings XML file that you selected.

8. Click Finish. The message "Factory Settings creation completed successfully" displays.

8.4.5.5.7 Importing Advanced XML Conditions from the Database

You can import existing advanced XML conditions or Greex rules from the database on to your local machine. This is useful if you want to modify the structure of an advanced XML condition. For example, when you want to add some new expressions or IF and ELSE constructs to an existing advanced XML condition.

To import advanced XML conditions from the database:

1. Start Eclipse SDK.

2. From the menu bar, select Window > Show View > Navigator. The java projects display in the Navigator view.

3. Right-click on the java project in which you want to import the `*.greex` files from the database. Select Greex > Import Greex Rule from Database from the pop-up menu. The Import Greex Rule to Database window displays. Each `*.greex` file contains an advanced XML condition or Greex rule. For more information about creating a java project, see Section 8.4.5.5.2, "Creating a Java Project".
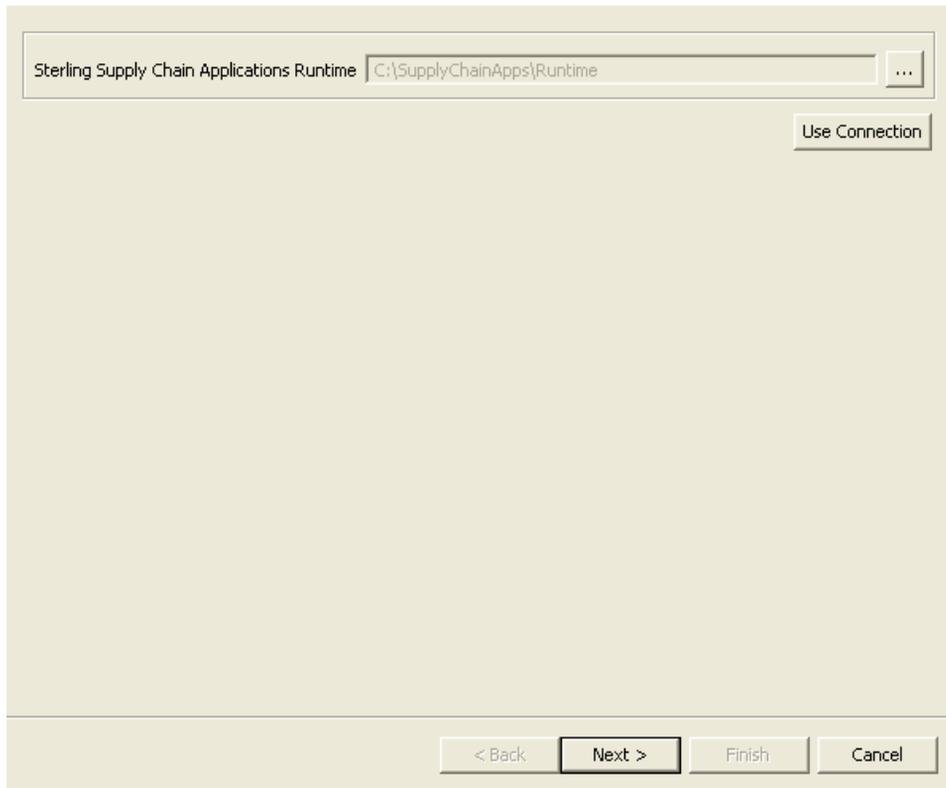
Table 8–19   Import Greex Rule from Database

Field	Description
Sterling Supply Chain Applications Runtime	Click [...] and browse to the `<YANTRA_HOME>/Runtime` directory. where <YANTRA_HOME> refers to the directory where you have installed the Sterling Supply Chain Applications. The Greex rules are fetched from the database specified in the `<YFS_HOME>/Runtime/resources/yfs.properties` file. where <YFS_HOME> refers to the `<YANTRA_HOME>/Runtime` directory.
Use Connection	Click this button to use the JDBC connection specified in the `<YFS_HOME>/Runtime/resources/yfs.properties` file. where <YFS_HOME> refers to the `<YANTRA_HOME>/Runtime` directory.

4. Click Next. The Select Greex Rules window displays.
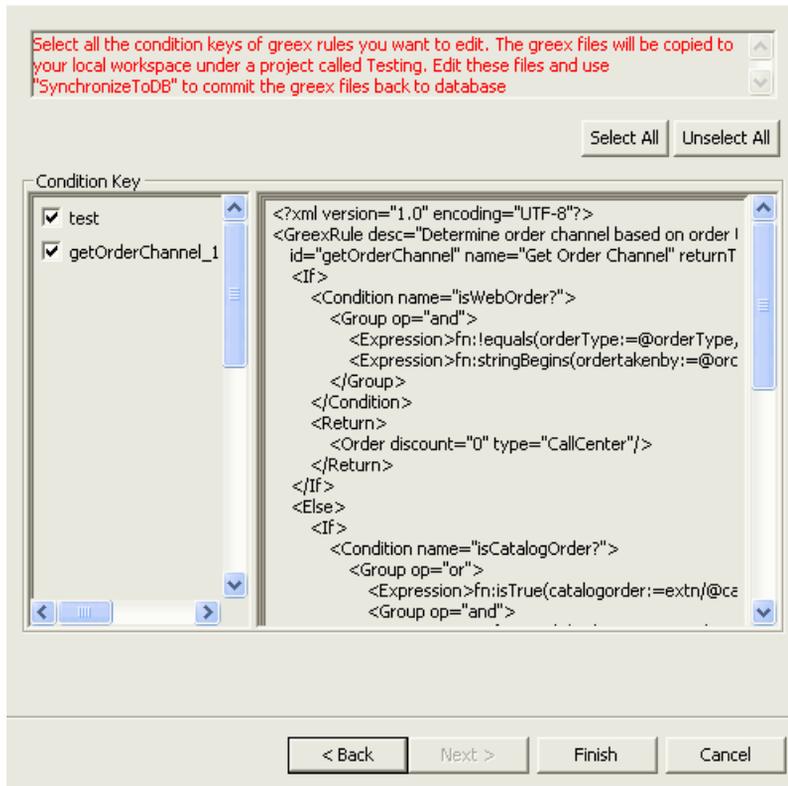
Table 8−20   Select Greex Rules Window

Field	Description
Select All	Click this button if you want to select all the Greex files listed in the Condition Key panel.
Unselect All	Click this button if you want to unselect all Greex files listed in the Condition Key panel.

Table 8–20   Select Greex Rules Window

Field	Description
Condition Key	
Displays the list of condition identifiers specified for each Greex file loaded in the database. If you place the mouse on a particular condition identifier, the right-hand panel displays the contents of the selected advanced XML condition or Greex rule. For example, the Greex rule or advanced XML condition identifier, name, description, and so forth, and all IF and ELSE constructs defined for that advanced XML condition displays. Unselect Greex files that you do not want to copy on your local machine.	

5. Click Finish. The selected Greex files are imported into the java project.

6. Open the Greex file using the Sterling Greex Editor and modify the advanced XML condition or Greex rule as needed.

7. Load the modified Greex files again into the database. For more information about loading the advanced XML conditions into the database, see Section 8.4.5.5.6, "Loading Advanced XML Conditions into the Database".

8.4.5.6 Modifying an Advanced XML Condition

Using the Sterling Supply Chain Applications Configurator, you can only change the modifiable parameters of an advanced XML condition. You cannot change the structure of an advanced XML condition. For more information about modifying advanced XML conditions, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

8.5 Dynamic Configuration Using Variables

You can dynamically configure certain properties belonging to actions, agents and services configuration to cut back on your implementation time. For example, you can provide a dynamic way of configuring network properties such as provider URLs, E-mail server, and Sender addresses. These properties can be configured as variables in one place and can be resolved at runtime when these properties are being used.

You can provide variables in place of file or directory names wherever a file name or path can be entered in the Sterling Supply Chain

Applications Configurator. This variable substitution is based on an entry in the `yfs.properties` file and is resolved during runtime.

You can use variable names in the following components of Sterling Supply Chain Applications Configurator:

- Sterling Service Definition Framework
    - All transport types such as JMS queue name, Initial Context Factory, QCF lookup and Provider URL. File Sender and Receiver, FTP source and destination for sender and receiver directories and HTTP and Webservice transport types.
    - In the e-mail component, where you can specify, the email server, subject, listener port, and From addresses. The dynamic configuration can also be used for specifying the email protocol.
    - Actions
        * In call HTTP extension and Execute Program.
    - Agent criteria details
        * JMS Queue Name, Initial Context Factory, QCF Lookup and Provider URL.
- Printer Devices, Print Documents, and Print Components
- Purge Criteria's log file name
- In Sterling Supply Chain System Management console:
    - You can use variables to specify installation rules such as e-mail server name, server IP address, server listener port, and e-mail protocol.
    - You can provide variables for the JMS monitoring configuration fields, which include: WebLogic Provider URL, WebSphere Channel name, host name, port number and queue manager name.

For the fields identified above, you can define the values as ${VARIABLE_NAME} in `<YANTRA_HOME>/Applications/Foundation/resources/yfs.properties` file. It is stored in the database as is and at runtime when the variables are used, a lookup is performed in the `yfs.properties` file to decipher the value. Since, the values for these variables are fetched from the `yfs.properties` file, they are specific to a particular JVM.

> **Note:** The value of this variable cannot be seen in the health monitor agent details, since the value depends on the JVM on which it is deployed. You have to click on the server details of the monitor agent to view the value of the variable.

For example, if you want to set the File IO Receiver's directory structure to a common variable say `ffbase`, then the incoming directory should be set to `${ffbase}/incoming`. The variable `ffbase` must be defined in the file as:

```
ffbase=C:/FileIODir/Receiver
```

This `${ffbase}/incoming` value is stored in the database, and when processing the file adapter, the variable is resolved to `C:/FileIODir/Receiver/incoming`.

The following conditions are assumed for the usage of this variable:

- All the variables when referenced must be in the following format:
  - ${variable_name}
- All variables should be properly formed. If a variable is not found, no substitution takes place.
- Variables must not contain the '}' character.
- Variables must not begin or end with a whitespace.
- Templates do not support variables for filenames since they are always resolved within the classpath.

8.6 Creating Custom Transactions

You can create the following custom transactions:

- Custom APIs and services
- Time-triggered Transactions

8.7 Creating Extended APIs

Extended APIs are APIs that you provide; they are sometimes called custom APIs. You can use an extended API to invoke a Sterling Supply Chain Applications or third-party API, as well as to perform custom processing through the Sterling Service Definition Framework.

To invoke an extended API:

1. Code a class.

2. Code a function that has exactly two parameters of types YFSEnvironment and Document and ensure that the function returns a document.

   ```
   public Document <method-name> (YFSEnvironment env, Document doc)
   ```

3. Configure a service that contains an API node. When configuring an API node, use the properties described in Table 8–21.

Table 8–21   API Node Configuration Properties

Property	Description
General Tab	
Extended API	Select this option if a custom API is to be invoked.
API Name	Select or enter the API to be called.
	Note: This field is for integration purposes only.
Class Name	Specifies the class you coded in Step 1.
Method Name	Specifies the function to be called as coded in Step 2.
Arguments Tab	
Argument Name	You can pass name/value pairs to the API by entering the values in the Arguments Tab.
	In order for custom APIs to access custom values, the API should implement the interface `com.yantra.interop.japi.YIFCustomApi`.
	If entered, these name/value pairs are passed to the custom API as a properties object.
Argument Value	Enter the argument value.

When connecting the nodes within a service, keep in mind the API node connection properties as listed in Table 8–22:

Table 8–22   API Node Connection Properties

Connection	Node Connection Rules
Can be the first node after the start node	Only for services invoked synchronously
Can be placed before	• Any transport node except FTP or File IO • Any other component node
Can be placed after	• Start node • Any transport node except FTP or File IO • Any other component node
Passes data unchanged	Yes

4. Make sure the class is in the CLASSPATH of the Sterling Service Definition Framework.

5. Make sure that the class implements a method with a signature that takes in exactly two parameters, a YFSEnvironment and a Document.

   The following example shows how to implement a class:

   ```
   import com.yantra.yfs.japi.YFSEnvironment;
   import org.w3c.dom.Document;
   public class Bar {
       public Bar () {
           }
       public Document foo(YFSEnvironment env, Document doc)
           {
           //write your implementation code here
           }
   }
   ```

6. To access the extended API you created, invoke the service containing your extended API.

   For details and sample code that show how to access properties specified when the custom API is configured, see the YIFCustomAPI interface in *Sterling Supply Chain Applications Javadocs*.

8.7.1 Implementing the Error Sequence User Exit

You can configure the Sterling Service Definition Framework to call a user exit that checks for prior errors for the exception group to which the API belongs. This user exit is called before any processing of the message starts. A Java interface is supplied for its implementation. This interface definition is in the `com.yantra.interop.japi.YIFErrorSequenceUE class`. The user exit computes the Message Key based on user defined custom code.

`YIFErrorSequenceUE` defines two functions. The function definitions are:

```
1) public Document getExceptionGroupReference(Document document, String apiName)
throws Exception
2) public void setExceptionGroupFinder (YIFExceptionGroupFinder finder)
```

The `getExceptionGroupReference()` function takes two parameters:

- Document - The input XML document retrieved by the Integration Adapter

- String - The API for which the Integration Adapter retrieved the XML

The `setExceptionGroupFinder()` function sets the `YIFExceptionGroupFinder()` interface. Use the implementation of this interface to retrieve the `exceptionGroupId` if prior errors exist.

An example implementation of this function is:

```
public void setExceptionGroupFinder (YIFExceptionGroupFinder finder){
 this.finder = finder;
}
```

8.7.2 Implementing the YIFExceptionGroupFinder Interface

This interface defines the `findExistingError()` function that takes in `Document` as the input parameter.

For example, the input XML document that the user exit passes to the `findExistingError()` function would contain:

```
<?xml version="1.0"?>
<ExceptionGroupReference messageKey="xyz"/>
```

8.7.3 Exception Handling in Extended APIs

The client always has the option of throwing an exception to the Sterling Service Definition Framework instead of handling it when it occurs. Depending on the configuration, the Sterling Service Definition Framework either sends the exception to the Alert Console or logs the exception.

8.8 Creating Custom Time-Triggered Transactions

Sterling Supply Chain Applications provide infrastructure that enables you to write your own time-triggered transactions. You invoke and schedule these time-triggered transactions in much the same manner as you invoke and schedule Sterling Supply Chain Applications standard time-triggered transactions. For information on how to configure Sterling Supply Chain Applications standard time-triggered transactions, see the *Sterling Supply Chain Applications Platform Configuration Guide*.

Depending upon the way time-triggered transactions determine the list of tasks to be processed (their work load), they can be classified into one of the following categories:

- Non task-based (generic) - these time-triggered transactions use custom logic to determine the work they have to perform. They may or may not use the centralized Task Queue.

- Task-based (specific) - these time-triggered transactions use the Task Queue to determine their work.

Sterling Supply Chain Applications provide infrastructure to create both types of custom time-triggered transactions.

The ability to write non task-based time-triggered transactions is provided using the `com.yantra.ycp.japi.util.YCPBaseAgent` class. This class provides a generic infrastructure irrespective of whether the Task Queue is used or not and therefore can be used to write any time-triggered transaction.

Task-based time-triggered transactions can be programmed by subclassing the `com.yantra.ycp.japi.util.YCPBaseTaskAgent` class.

If your transaction is Task Queue based, it is suggested that you use the infrastructure provided specifically to write task-based transactions. This

infrastructure automatically determines work for your custom agent from the Task Queue, thus reducing the amount of design and development required for your transaction.

All custom agents written to Sterling Supply Chain Applications specification are subclassed from the `com.yantra.ycp.japi.util.YCPBaseAgent` class, which has two abstract functions. When you implement these functions, they provide the processing capabilities of a time-triggered transaction.

getJobs() Abstract Function

The `getJobs()`abstract function uses `Env`, a pre-created instance of a `YFSEnvironment` object that can be passed to APIs and `inXML` is an `org.w3c.dom.Document` object, which contains the input XML. The implementation of this function should obtain jobs (from the database) that need to be executed, construct a list of org.w3c.dom.Document objects, and return the list. See the following signature:

```
public List getJobs(YFSEnvironment Env, Document inXML)
```

executeJobs() Abstract Function

Each document in the List returned by the `getJobs()` function is passed to this function for execution. If this function throws an exception, the exception is logged and the transaction is rolled back, otherwise it is committed. See the following signature:

```
public Document executeJob(YFSEnvironment Env, Document inXML)
```

After all of the documents have been processed by the `executeJob()` function, Sterling Supply Chain Applications invoke `getJobs()` function again to obtain the next set of tasks that need to be processed by the `executeJob()` function. This repeats until no more jobs are returned by the `getJobs()` function.

For examples of the input XML, see the YCPBaseTaskAgent class in the *Sterling Supply Chain Applications Javadocs*.

The `com.yantra.ycp.japi.util.YCPBaseAgent` also provides utility functions for trace logging and timer information with the following signatures:

- public void log(String classMethodName, String message);
- public startTimer(String timerName);

- public endTimer(String timerName);

8.8.1 Creating Custom Non Task-Based Time-Triggered Transactions

Custom non task-based time-triggered transactions should be written as subclasses of the `com.yantra.ycp.japi.util.YCPBaseAgent` class. For a sample custom time-triggered transaction, see the `com.yantra.ycp.japi.util.YCPBaseAgent` class in *Sterling Supply Chain Applications Javadocs*.

To write a custom non-task based time-triggered transaction:

1. Subclass `com.yantra.ycp.japi.util.YCPBaseAgent`.

2. Implement the `executeJob()` and `getJobs()` functions in this class.

3. From the Configurator, configure a time-triggered transaction and assign an Agent Server to it. For details about configuring time-triggered transactions see the *Sterling Supply Chain Applications Platform Configuration Guide*.

4. Schedule and run your custom time-triggered transaction according to the instructions in the *Sterling Supply Chain Applications Installation Guide*.

8.8.2 Creating Custom Task-Based Time-Triggered Transactions

Task-based custom time-triggered transactions are written as subclasses of the `com.yantra.ycp.japi.util.YCPBaseTaskAgent` class, which is a subclass of `com.yantra.ycp.japi.util.YCPBaseAgent` with the `getJobs()` and `executeJob()` functions already implemented. Creating a task-based custom time-triggered transaction by subclassing this class involves implementing the `executeTask()` function to process one task queue record passed to you as input.

> **Note:** If the `executeTask()` function throws an exception, the exception is logged and the transaction is rolled back, otherwise it is committed.

The logging and timing utility functions available are similar to the ones provided by the `com.yantra.ycp.japi.util.YCPBaseAgent` class. The signature of the `executeTask()` function is `public Document executeTask(YFSEnvironment oEnv, Document inXML);Env` is a pre-created instance of a `YFSEnvironment` object that can be passed to APIs and `InXML` is the `org.w3c.dom.Document` object, which contains the custom task XML. The custom task XML also contains a TransactionFilters Node, which contains all the parameters passed to the task-based custom time-triggered transaction. This node is below the root node of the input XML. For example, see an example for a task-based custom time-triggered transaction.

```
<?xml version="1.0" encoding="UTF-8"?>
<TaskQueue TaskQKey="" TransactionKey="" DataKey="" DataType="" AvailableDate=""
    Lockid="" Createts="" Createprogid="" Createuserid=""
Modifyts="" Modifyprogid="" Modifyuserid="" >
            <TransactionFilters   AgentName=""  TransactionKey=""
         CurrentThread="" NumRecordsToBuffer="" TotalThreads=""/>
</TaskQueue>
```

For a sample custom task-based time-triggered transaction, see the `com.yantra.ycp.japi.util.YCPBaseTaskAgent` class in *Sterling Supply Chain Applications Javadocs*.

To write a task-based custom time-triggered transaction:

1. Subclass `com.yantra.ycp.japi.util.YCPBaseTaskAgent`.

2. Implement the `executeTask()` function in this class.

3. From the Configurator, configure a time-triggered transaction and assign an Agent Server to it. For details about configuring time-triggered transactions see the *Sterling Supply Chain Applications Platform Configuration Guide*.

Schedule and run your custom time-triggered transaction according to the instructions in the *Sterling Supply Chain Applications Installation Guide*.

8.9 External Transaction Coordination

Sterling Supply Chain Applications provide capability to execute external code either by raising an event, calling a user exit, or invoking a custom API or service. During these invocations, the external systems can begin a transaction. Since the external transaction is not part of the Sterling Supply Chain Applications transaction it could lead to data inconsistencies if the Sterling Supply Chain Applications transaction is rolled back but not the external transaction.

External transaction coordination lets the external systems register their transactions with Sterling Supply Chain Applications. When Sterling Supply Chain Applications are ready to commit its transaction, the YFSTxnCoordinatorUE user exit is invoked which lets Sterling Supply Chain Applications handle the commits and rollbacks on the external transactions.

To implement external transaction coordination:

1. Create the custom API that you want to implement your external transaction process, but DO NOT commit the external transactions. Instead, register the external transaction object with the YFSEnvironment by calling the (YFSEnvironment)env.setTxnObject(String ID, Object txnObj).

   The String ID is a unique name used by the user exit implementation class to identify the custom transaction object.

   The following example illustrates a simple custom API.

Example 8–10 External Transaction Coordination of a Custom API

```
public class doSomethingAPI
{
private YFCLogCategory cat = YFCLogCategory.instance("DoSomethingAPI");

public doSomethingAPI()  // constructor is empty
{
}
public void writeToDB (String key, YFSEnvironment oEnv)
{

try
{
Driver aDriver =
(Driver)Class.forName("oracle.jdbc.OracleDriver").newInstance();
```

```
          String url = "jdbc:oracle:thin:@127.0.0.1:1521:qotree2";
          Connection conn = DriverManager.getConnection(url, "Scott",
"Tiger");
conn.setAutoCommit(false);
          String sql = "insert into TxnTest (key) values ('" + key + "')";
          Statement stmt = conn.createStatement();
          stmt.executeUpdate(sql);
          oEnv.setTxnObject("YDBconn", conn);
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
     }
public Document doSomething(YFSEnvironment env, Document doc) throws
Exception
{
      System.out.println("Executing doSomething method...........");
writeToDB ("doSomething", env);
return doc;
     }
}
```

2. Implement the `com.yantra.yfs.japi.ue.YFSTTxnCoordinatorUE`
   user exit interface to commit the external transactions either before
   or after Sterling Supply Chain Applications perform their commits.
   Then implement the rollback method so that if the Sterling Supply
   Chain Applications transaction triggers a rollback, the
   `afterYantraTxnRollback(YFSEnvironment oEnv)` method is called
   to rollback the external transactions as well. Implement the following
   methods to accomplish this:

   - `beforeYantraTxnCommit(YFSEnvironment oEnv)`

   - `afterYantraTxnCommit(YFSEnvironment oEnv)`

   - `afterYantraTxnRollback(YFSEnvironment oEnv)`

> **Note:** You can use either the `beforeYantraTxnCommit` or the `afterYantraTxnCommit` user exit to synchronize commits, depending on your integration requirements.

Calling `(YFSEnvironment)env.getTxnObject(ID)` enables these methods to obtain the handle to the external transaction object that was previously registered by the `(YFSEnvironment)env.setTxnObject(String ID, Object txnObj)`. Note the ID is the same in both the `getTxnObject` call and the `setTxnObject` call.

The following is an example of the `YFSTTxnCoordinatorUE` user exit interface implementation.

Example 8–11   Sample Interface Implementation

```
public class afterTxnCommit implements YFSTxnCoordinatorUE
{

public void beforeYantraTxnCommit (YFSEnvironment oEnv) throws
YFSUserExitException
{
// before method is not implemented because after method is implemented.
}
public void afterYantraTxnCommit (YFSEnvironment oEnv) throws
YFSUserExitException
{
System.out.println ("Entering method afterYantraTxnCommit.........");
try
{
    Connection ydbConn = (Connection)oEnv.getTxnObject("YDBconn");
    ydbConn.commit();
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}
}
public void afterYantraTxnRollback (YFSEnvironment oEnv) throws
YFSUserExitException
{
System.out.println ("Entering method afterYantraTxnRollback.........");
```

```
try
{
    Connection ydbConn = (Connection)oEnv.getTxnObject("YDBconn");
    ydbConn.rollback();
}
catch (Exception e)
{
System.out.println ("Caught Exception :\n" + e);
}

}
```

3. Launch the Sterling Supply Chain Applications Configurator and navigate to System Administration > User Exit Management to configure the `YFSTxnCoordinatorUE` Implementation class as described in the *Sterling Supply Chain Applications Platform Configuration Guide*.

For more information about the `YFSTxnCoordinatorUE` user exit interface definition and other detailed information, see the *Sterling Supply Chain Applications Javadocs*.

8.10 Invoking APIs and Services

You can invoke APIs and Services from the client environment or programmatically.

8.10.1 Invoking APIs from the Client Environment

In order to call standard APIs from the client, ensure that the client environment is set up correctly. The client environment must have appropriate CLASSPATH settings and the JAR files as described in this section.

The `<YANTRA_HOME>/Applications/Foundation/resources/` directory must contain the `yifclient.properties` file.

If you are calling in local mode, the client CLASSPATH must contain the following files of Sterling Supply Chain Applications:

- `yompbe.jar`
- `yimbe.jar`
- `yfcbe.jar`

- `ydmbe.jar`
- `ycsbe.jar`
- `ycpbe.jar`
- `ycmbe.jar`
- `yantraiconsbe.jar`
- `wmsbe.jar`
- `vasbe.jar`
- `yantrashared.jar`
- `yfsclient.jar`
- `log4j-1.2.11.jar`
- `yfcbe.jar`
- `xercesImpl.jar`
- `xml-apis.jar`

When running on WebSphere in EJB or HTTP mode, the client CLASSPATH should contain the following WebSphere files in `<WAS_HOME>/properties` directory:

- `yfsclient.jar`
- `log4j-1.2.11.jar`
- `yfcbe.jar`
- `xercesImpl.jar`
- `xml-apis.jar`

When running on the WebLogic application server in EJB mode, the client CLASSPATH should have the `weblogic.jar` WebLogic file.

When running on IBM JDK 1.4.* prepend the xercesImpl.jar, xml-apis.jar and xalan.jar files to the bootclasspath. For example in UNIX, the classpath is prepended as follows:

```
-Xbootclasspath/p:<YANTRA_
HOME>/Applications/Foundation/lib/xercesImpl.jar:<YANTRA_
HOME>/Applications/Foundation/lib/xml-apis.jar:<YANTRA_
HOME>/Applications/Foundation/lib/xalan.jar
```

8.10.2 Invoking Services and APIs

Sterling Supply Chain Applications provide sample code that demonstrates how the Sterling Supply Chain Applications standard APIs and services can be invoked programmatically. See the sample files in the `<YANTRA_HOME>/Applications/Foundation/documentation/code_examples/` directory.

> **Note:** Use the `executeFlow()` method of the YIFApi interface to execute a service defined within the Sterling Service Definition Framework.

When calling APIs in LOCAL mode, copying the management.properties file to your local `<YANTRA_HOME>/Applications/Foundation/resources/` directory enables the database cache to refresh when required. For more information about the `managment.properties` file, see the *Sterling Supply Chain Applications Installation Guide*.

API and service transactions that are outbound from Sterling Supply Chain Applications can be configured through the Service Builder, as described in the *Sterling Supply Chain Applications Platform Configuration Guide*.

API and service transactions that are inbound to Sterling Supply Chain Applications can be invoked through the following protocols:

- EJB
- HTTP and HTTPS
- LOCAL
- Web Services
- COM+

EJB

Use EJB for server-side execution of code. Java call. All the methods in Sterling Supply Chain Applications take a `YFSEnvironment` and a `document` and returns a `document`. Since EJBs are designed to be called remotely, each of these documents is serialized on one end and unserialized on the other. However, Sterling Supply Chain Applications

use an EJB, where each API takes two string parameters and returns a string. Thereby, forcing any document implementation to serialize and unserialize using a standard well-defined interface.

For example, a new EJB is created with method signatures like:

```
String createOrder(String env, String inputXML) throws
YFSException, RemoteException;
```

where `env` is an XML that should be a valid input to `createEnvironment` variable. The return value is the output XML.

When calling an API using
`YIFClientFactory.getInstance().getApi("EJB")` the call is made using this String-based EJB. With this type of call you can pass a `YFSEnvironment` and `document`, and get a `document` in return. The Sterling Supply Chain Applications code performs the conversion transparently.

> **Note:** The DOM-based EJB is deprecated. Hence, moving forward you need to use the String-based EJB for server-side execution.

HTTP
Use HTTP for server side execution of code. Java call.

LOCAL
Use Local for client side execution of code. COM **or** Java call.

Web Services
Use Web Services for client side execution of code. COM **or** Java call.

COM+
Use COM for client side execution of VB or C++ code. COM **or** Java call.

Using COM requires setting up your server and runtime clients as described in "Using Sterling Supply Chain Applications with Microsoft COM+" on page 16.

> **Note:** Exceptions encountered when making synchronous API calls through EJB, COM, or HTTP transport protocols are not queued for reprocessing.

To configure service invocation:

1. Copy the `<YANTRA_ HOME>/Applications/Foundation/resources/yifclient.propertie s.sample` file to `<YANTRA_ HOME>/Applications/Foundation/resources/yifclient.propertie s`.

2. Set your CLASSPATH to include the directory containing the resources directory and the following files:

   - log4j-1.2.11.jar
   - xercesImpl.jar
   - yfcbe.jar
   - yfsclient.jar

3. Set your java command line property to `-Dlog4j.configuration=resources/log4jconfig.xml`.

4. Make sure that the `<YANTRA_HOME>` directory is in your CLASSPATH.

5. Set the log4j properties in the `log4jconfig.xml` file to the appropriate values for your environment. If these properties are not specified correctly, the Sterling Service Definition Framework does not initialize correctly.

   - If you are using the EJB protocol and *WebLogic*, make sure that `weblogic.jar` is in your CLASSPATH environment variable. In addition, `xercesImpl.jar` and `xalan.jar` must precede `weblogic.jar` in your CLASSPATH.

   - If you are using the EJB protocol and *WebSphere*, make sure that the CLASSPATH environment variable contains the following JAR files and directories:

* `<WAS_HOME>/lib/namingclient.jar`

* `<WAS_HOME>/lib/runtime.jar`

* `<WAS_HOME>/lib/messaging.jar`

* `<WAS_HOME>/installedApps/yantra.jar`

* `<WAS_HOME>/properties/`

— If you are configuring a COM+ protocol call, use one of the following COM signatures that you need:

```
createEnvironment(VARIANT *lEnvHandle, BSTR sProgID, BSTR sUserID, int
*iRetval)
```

Signature for calling standard APIs:

```
<StandardYantraApi>(VARIANT *lEnvHandle, BSTR inXML, VARIANT *outXML,
VARIANT *errXML, int *retval)
```

Signature for calling services:

```
executeFlow(VARIANT *lEnvHandle, BSTR flowName, BSTR flowMsg, VARIANT
*outXML, VARIANT *errXML, int *retval)
```

For examples of VB code, see the samples in the `<YANTRA_ HOME>/Applications/Foundation/documentation/code_ examples/complus` directory.

8.11 Transactional Data Security

Security issues involve controlling who has access to what data, how much they can see, and what they can do with it. Sterling Supply Chain Applications provide mechanisms that address the following security issues:

* User access control

* Single sign on

* Data encryption

8.11.1 User Access Control

Through an access control mechanism of user group permissions, logins, and order modification permissions, Sterling Supply Chain Applications

provide security measures for multiple levels of customer service and administrative organizations.

APIs control access to different areas of system functionality. It is the responsibility of the caller to ensure that the invoking user has access rights for the function being invoked. Sterling Supply Chain Applications provide security manager APIs to help in this effort. See the `com.yantra.api.ycp.security` package in the *Sterling Supply Chain Applications Javadocs.*

8.11.2 Single Sign On

If a single sign on mechanism is required within the Sterling Supply Chain Applications, implement the interface `com.yantra.ycp.japi.util.YCPSSOManager`, which has a `getUserData()` function that returns a `String(UserId)`. For more detailed information, see the *Sterling Supply Chain Applications Javadocs.*

8.11.3 Data Encryption

Encryption ensures that sensitive data is not viewed by unauthorized people. Sterling Supply Chain Applications provide APIs that enable you to encrypt data such as user names, passwords, and credit card numbers.

In addition, encryption and decryption is only applied after it has been specified within the Configurator. For example, only user exits that have been passed credit card information can access decrypted credit card numbers.

8.11.4 Encryption Logic in Sterling Supply Chain Applications

Sterling Supply Chain Applications expose `com.yantra.ycp.japi.util.YCPEncrypter` interface to handle encryption logic. All of Sterling Supply Chain Applications encryption and decryption is handled by an encrypter class that implements this interface. This class is specified by the `yfs.encrypter.class` and `yfs.propertyencrypter.class` properties in the `yfs.properties` file. Both classes must implement the `com.yantra.ycp.japi.util.YCPEncrypter` interface.

The `com.yantra.ycp.japi.util.YCPEncrypter` interface has the following two functions:

- public java.lang.String encrypt(java.lang.String sData) - sData is the data passed by Sterling Supply Chain Applications to the implementing class for encryption. The return value is the encrypted string.

- public java.lang.String decrypt(java.lang.String sData) - sData is the data which is required to be decrypted.

For information on writing your own property encrypter class, see the YCPEncrypter interface in the *Sterling Supply Chain Applications Javadocs*.

Encryption and decryption functions in this interface are invoked multiple times by Sterling Supply Chain Applications. Sterling Supply Chain Applications do not distinguish between clear text and encrypted information. Therefore, the encrypt function may be invoked with previously encrypted data. In order to avoid double encryption, it is important for the encrypt function to be able to distinguish between clear text and previously encrypted information. If previously encrypted information is passed to the function, your implementation of this function should return what is passed into it without encrypting it again.

The decrypt function also should be able to distinguish between clear text and previously encrypted text.

8.11.5 Disabling Encryption and Decryption

To disable encryption (or decryption), implement the encrypt (or decrypt) function to return the same value it is passed as input without any processing.

8.11.6 Choosing a Deployment Strategy

There are multiple deployment options when choosing an encryption strategy. The most typical options are:

- No encryption or decryption
- Both encryption and decryption
- No decryption

Use the following explanation to guide your decision-making process:

No Encryption or Decryption

If you operate in a secure and trusted environment which is protected physically and electronically and you do not display credit card numbers on the Sterling Supply Chain Application Consoles, you may choose not to implement any encryption logic. Credit Card numbers are be encrypted in this case and are stored in clear text. This is not a recommended option except in the following scenarios:

- Your business does not accept, process, or store credit card numbers or other sensitive information.

- Sterling Supply Chain Applications are always passed externally encrypted credit card numbers. All encryption and decryption is handled externally.

Both Encryption and Decryption

Sterling Supply Chain Applications encrypt and decrypt credit card numbers automatically as required. Access to clear text credit card numbers is available on the Sterling Supply Chain Application Consoles based on user authorization levels.

No Decryption

If your business requires Sterling Supply Chain Applications to store credit card numbers, but you never want Sterling Supply Chain Applications to automatically decrypt them under any circumstances, you may want to enable only the encrypt function and disable the decrypt function.

This way, Sterling Supply Chain Applications encrypt credit card numbers passed in as clear text but never converts them back. Once Sterling Supply Chain Applications encrypt information, all your custom extensions are passed encrypted credit card numbers and must handle decryption externally. It is important to note that a few user exits in Sterling Supply Chain Applications (for example, `YFSbeforeCreateOrderUE`) are invoked *before* the credit card number is encrypted, so it still has access to the clear text number.

8.11.7 Encryption Usage in Sterling Supply Chain Applications

Sterling Supply Chain Applications support encryption for the following places:

- Properties specified in the yfs.properties, yif.properties, management.properties files

- Credit Card Numbers

Properties Specified in the yfs.properties File

Properties such as the JDBC URL, database User ID and Password can be stored encrypted in the `yfs.properties` file. Because Sterling Supply Chain Applications need this information to connect to the database, these values must be decrypted by Sterling Supply Chain Applications. If you do not wish Sterling Supply Chain Applications to ever decrypt data, these properties cannot be stored encrypted.

Credit Card Numbers

Sterling Supply Chain Applications can encrypt Credit Card numbers before storing them in the database. Unlike the properties specified in the `yfs.properties` file, decrypted credit card numbers are never required by Sterling Supply Chain Applications for default processing. However, you may extend Sterling Supply Chain Applications by implementing a user exit that requires decrypted credit card numbers for charging or storing user preferences. If you don't want Sterling Supply Chain Applications to decrypt information automatically, you must decrypt these credit card numbers in your implementation of the user exit.

8.11.8 Encrypting Credit Card Numbers

If you implement encryption, Sterling Supply Chain Applications encrypt credit card number in the following situations:

- When returned by an API

- When published as a part of event data

- When stored anywhere in the database

- When displayed on the user interface (although the user interface may have an option to override this behavior based on user access)

Credit Card Encryption APIs

Sterling Supply Chain Applications provide following APIs that enable you to encrypt and decrypt credit card numbers assuming that both encryption and decryption algorithms have been implemented by the Encrypter class:

- getEncryptedCreditCardNumber() - returns an encrypted credit card number
- getEncryptedString() - accepts a string passed to it and returns the string encrypted
- getDecryptedCreditCardNumber() - returns the credit card number that had been encrypted using the getEncryptedCreditCardNumber() API
- getDecryptedString() - accepts an encrypted string and returns the string decrypted

Credit Card Encryption User Exits

Only user exits that are passed credit card information can access decrypted credit card numbers. Sterling Supply Chain Applications provide following user exits for passing credit card data. For detailed information about these user exits, see the *Sterling Supply Chain Applications Javadocs.*

8.11.9 Encrypting Properties

Some properties relay sensitive data such as user IDs and passwords, which you may want to encrypt. Any property (except for the `yfs.propertyencrypter.class` property in the yfs.properties file), can be encrypted as needed within the following files:

- yfs.properties
- yif.properties
- management.properties

To encrypt properties

When encrypting properties, you need to:

1. Append the property you want to encrypt with ".encrypted". Note that you cannot encrypt the yfs.propertyencrypter.class property.

2. Ensure that the yfs.propertyencrypter.class property is accessible through the CLASSPATH environment variable

3. Implement the YCPEncrypter interface. For details about this interface, see the *Sterling Supply Chain Applications Javadocs*.

These properties ending with .encrypted are automatically decrypted at run-time.

8.12 Deploying

> **Note:** At times, mechanisms supplied by Sterling Supply Chain Applications such as time-triggered transactions, APIs, and user exits are replaced by improved mechanisms. When these mechanism are replaced, they are designated as "deprecated." Whenever possible, use the new mechanisms rather than the deprecated ones.
>
> If you do need to use a deprecated mechanism, it must be run in backward compatibility mode. In addition, note that deprecated mechanisms are supported for two major software versions, and then they are removed from the product.

When you create any custom code (user exits, extended APIs, custom implementations of supplied Java interfaces, and so forth), you must add them to a JAR file in the <YANTRA_ HOME>/Applications/Foundation/extn/ directory and the yantra.ear file must be rebuilt and re-deployed as described in the *Sterling Supply Chain Applications Installation Guide*.

A

Special Characters Reference

Sterling Supply Chain Applications reserve keywords and special characters that are only used internally. This appendix details the use and handling of the special characters in the following situations:

- Naming Files - see page 561

- Avoiding Sterling Supply Chain Applications Reserved Keywords - see page 562

- Forming API Input - see page 562

- Using Multi-Byte Characters - see page 564

For more details about using special characters, see the *Sterling Supply Chain Applications Localization Guide*.

A.1 Naming Files

When naming files, Sterling Supply Chain Applications recommend that you choose characters from the standard English-based character set, such as A through Z and 0 (zero) through 9. That way, if you need to localize the application in languages other than English, you do not need to rename files.

When creating files in order to extend the user interface, see the standards noted in Appendix C, "Console JSP Interface Extensibility Reference".

A.2 Avoiding Sterling Supply Chain Applications Reserved Keywords

Some keywords are reserved for use by Sterling Supply Chain Applications and are important to keep in mind when programming with APIs and creating error codes. Do not create file names or error codes that start with the following:

- DCS
- INV
- OMD
- OMP
- OMR
- OMS
- SYS
- WMS
- YCM
- YCP
- YDM
- YFC
- YFE
- YFS
- YFX
- YIF
- YRET

A.3 Forming API Input

When coding API input parameters, follow the guidelines in this section for using literals and formatting API input.

> **Caution:** Do not pass a blank element (an element containing all the attributes with blank values) to an API. Also, do not pass attributes that have leading or trailing spaces. The result of either situation is not predictable.

A.3.1 Using Literals in Maps and XMLs

Use literals in maps and XMLs. Using literals enables you to write code with fewer bugs because the compiler catches the use of incorrect names in the <name>=<value> pair. In addition, using literals simplifies the maintenance of your code; if you change the <name>, all you need to do is recompile your code instead of editing one or more <name>s within it first.

A.3.2 Using Special Characters

Fields that are part of a logical key for any record in the Sterling Supply Chain Applications schema (such as OrganizationCode and OrderNo) have specific restrictions. For these fields, Sterling Supply Chain Applications *do not* support use of the following special characters:

- Ampersand (&)
- Greater than symbol (>)
- Less than symbol (<)
- Percent sign (%)
- Quotation mark (")
- Plus sign (+)
- Apostrophe (')

> **Note:** The plus sign (+) is only allowed to be used in the ItemID field.

In addition, Sterling Supply Chain Applications advise avoiding the use of third-party vendors' reserved special characters. For example, in certain situations, data with underscore characters ("_") on an Oracle database could result in unexpectedly slow query performance because the database deciphers the underscore as a single character wild-card.

The following fields have no restrictions and support all characters:

- All description fields (for example, item description)
- All name fields (for example, organization name)
- All address fields (for example, billing address)

> **Note:** However when creating address fields through the UI, the information entered after the quotation mark (") is truncated and will appear as a new entry in `YFS_PERSON_INFO` table. To work around this problem, use apostrophe (') instead of quotations.

- All instruction fields (for example, gift wrapping)
- All text fields (for example, reasons and comments)

XML-Based APIs

The following table lists all of the following special characters that should follow the XML escape format.

Table A–1   Special Characters in Attributes of XML-Based APIs
Table A–2

For This Character	Enter This Sequence
quotation mark (")	"
single quotation (')	'
greater than symbol (>)	>
less than symbol (<)	<
ampersand (&)	&

A.4 Using Multi-Byte Characters

If you want to use multi-byte characters, your database must be configured to support multi-byte characters. For more information about multi-byte characters and localization, see the *Sterling Supply Chain Applications Localization Guide*.

B

RCP General Concepts Reference

This appendix covers the following topics:

- Rich Client Architecture
- Eclipse and Its Rich Client Platform
- Workbench
- Plugin Manifest Editor
- YRCPluginAutoLoader Extension Point
- Creating New Actions
- Registering a Plugin
- Registering Plugin Files
- Validating Controls
- Siblings
- RCP Utilities

B.1 Rich Client Architecture

Rich Internet Clients have advantages of both Client-Server and thin-client applications. Rich Internet Client applications are developed on open standards and have strong integration with the Desktop Operating System (OS), resulting in rich interaction. Rich Internet Client applications provide immediate feedback to users when they interact with the application. Rich Internet Client applications use modern UI controls, such as tree controls or tabbed panels. Also, Rich Internet Client applications allow users to perform interactive operations such as drag and drop.

User Interfaces (UI) have been an integral part of any software application. For the last few decades, a wide range of architectures and technologies have been used to deliver user interfaces. The Total Cost of Ownership (TCO) and Usability, Responsiveness, and Performance (URP) have been the two balancing factors for choice of technologies.

TCO covers all the upfront and ongoing costs of an application, which includes: purchase price, equipment, installation, training, and ongoing maintenance.

URP measures the performance of an application, its usability, and user's productivity.

The ideal application would be one with a low TCO and a high URP.

UI architectures can be classified as:

- Green screen (or Character User Interfaces (CUI))
- Client-Server
- Browser based
- Rich Internet Client

The CUI provided users with basic user interfaces. CUI did not have the capability of displaying information such as product images due to lack of graphic capabilities. With the advent of Graphical User Interfaces (GUI) and operating systems such as Windows, applications can support more sophisticated user interfaces along with alternate input devices. For example, mouse.

The GUI applications developed using Client-Server technologies resulted in Dynamic Link Library (DLL) conflicts and heavy network usage with respect to TCO.

In mid-90's, internet technologies such as Hyper Text Markup Language (HTML) started emerging very fast. Due to its simplicity and very low TCO, internet technologies had tremendous impact in the way business applications were delivered. Initially, HTML was only used for displaying information. However, its potential for applications was soon exploited.

HTMLs performance in an interactive mode is highly limited, with high number of trips required back and forth from the server. In addition, standard HTML was never intended to produce the high-quality and high-performance user interfaces that excelled under the Client-Server model.

The reduction in TCO of these browser-based applications was at the expense of the users, as the URP was severely reduced.

Figure B–1 illustrates the Rich Client Architecture.

Figure B–1   Rich Client Architecture
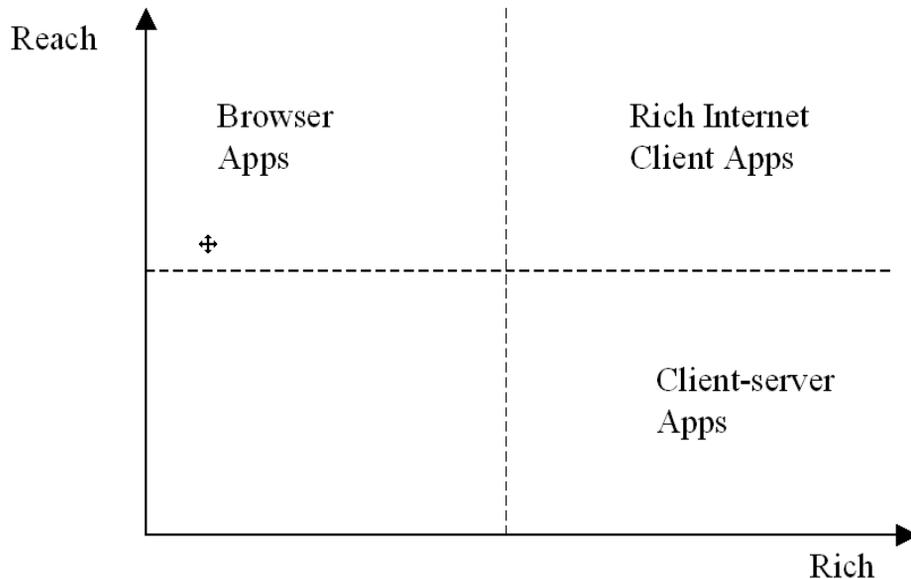
Today, technologies exists to create what is commonly known as Rich Internet Clients that have advantages of a Client-Server model in terms of URP and thin-client applications in terms of TCO. Rich Internet Client applications are developed on open standards and have strong integration with the desktop Operating System (OS), which results in highly rich interaction. Rich Internet Client applications are also designed to provide server-based updates and are designed to work with low bandwidth networks and standard security protocols.

B.2 Eclipse and Its Rich Client Platform

Eclipse is an open source software development environment dedicated to provide a robust, full-featured, and commercial-quality industry platform for the development of highly integrated tools. The Eclipse

Platform is designed for building Integrated Development Environments (IDEs) that are used to create diverse applications.

Eclipse RCP helps in building java applications that are platform independent. Eclipse RCP can also be used for building non-IDE applications. Eclipse RCP provides a general UI, which can be extended by developers to suit their business needs.

www.eclipse.org is an association of software development tool vendors. The Eclipse community was formed in order to create better development environments and product integration. The community shares an interest in creating products that are easily interoperable, because they are based on plugin technology and a common platform.

The Eclipse platform, which is a part of the Eclipse project, is an open extensible Integrated Development Environment (IDE). The Eclipse platform provides building blocks and a foundation for constructing and running integrated software development tools. Primarily, Eclipse platform is driven by International Business Machines (IBM). Eclipse technology is widely accepted within the Java community.

The Eclipse RCP addresses the need for a single cross-platform environment to create highly-interactive business applications. Essentially, RCP provides a generic Eclipse workbench that developers can extend to construct their own applications. Eclipse RCP is a part of Eclipse 3.2 release. Eclipse RCP enables application developers to deliver rich internet applications that run on platforms such as Windows, Linux, and so forth.

B.3 Workbench

Workbench refers to the desktop development environment. Workbench window contains one or more perspectives. A perspective defines the initial set and layout of views in the Workbench window. Perspectives contain views, editors, menus, and tool bars. You can customize a perspective by defining a set of actions. More than one Workbench window can exist on the desktop at any given time.

B.4 Plugin Manifest Editor

The Plugin Manifest Editor provides a single UI for editing the manifest and other plugin related files. The Plugin Manifest Editor contains following sections:

B.4.1 Overview

The Overview section provides plugin details such as plugin identifier, version, and so forth. It also specifies the class that is called when the user runs a plugin.

B.4.2 Dependencies

The Dependencies section provides a list of dependant plugins required by the plugin to compile its code. If a plugin is using the extension points of some other plugins, then the plugin must list those plugins as dependant plugins.

B.4.3 Runtime

The Runtime section provides a list of libraries in which the plugin code is packaged. For example, sop.jar. The class loader searches these libraries during runtime to load the plugin's classes. You can set the library's type, visibility, and content in the runtime section.

B.4.4 Extensions

The Extensions section describes the functionality that a plugin contributes to the Eclipse platform by extending other plugins extension points. The extension declaration must adhere to the schema defined by the extension point it extends. You can add new menus and menu items along with toolbar by extending the org.eclipse.ui.actionSets extension point.

B.4.5 Extension Points

The Extension Points section provides a list of new extension points that are defined by a plugin, which can be extended by other plugins to add the new functionality. For example, the Sterling RCP plugin provides a YRCPluginAutoLoader extension point which other plugins can extend to load their plugin.

B.4.6 Build

The Build section provides a list of libraries that are required at the runtime. It also lists the source folder where these libraries are located. You can select the folders and/or files you want to include in the source build and binary build.

B.4.7 Manifest.mf

The `manifest.mf` file contains a list of plugins that are loaded dynamically. The Bundle-Activator entry specifies the name of the plugin. For example, `com.yantra.yfc.rcp`.

B.4.8 Plugin.xml

The `plugin.xml` file contains all information that is required to run a plugin. The `plugin.xml` file is used for defining Eclipse extension points, and other dependent plugin's extension points. However, if you are not using any extension points, you can exclude this file.

B.4.9 Build.properties

The `build.properties` file contains all files and directories that are required by a plugin at the runtime.

B.5 YRCPluginAutoLoader Extension Point

The Sterling RCP provides "YRCPluginAutoLoader" extension point, which defines the order in which the plugins needs to be loaded. The "YRCPluginAutoLoader" is an extension point, which is defined in the "com.yantra.yfc.rcp" plugin. Any plugin that is dependent on the "com.yantra.yfc.rcp plugin" can extend this extension point to automatically load a class in the specified order when starting the Sterling Rich Client application. The "YRCPluginAutoLoader" automatically loads the classes within a plugin during startup in a specified order. All classes that need to be automatically loaded are sorted in ascending order and loaded one at a time. The "YRCPluginAutoLoader" has a extension element called AutoLoad, which has two properties "ClassToLoad" and "LoadOrder".

> **Note:** Loading a class within a plugin may load the plugin itself, resulting in initialization of the class used for registering plugin and other resource files. Therefore, the "YRCPluginAutoLoader" extension point is used for initialization purposes.

B.6 Creating New Actions

You can create new actions that can be invoked when you click on an menu item.

To create a new action:

1. Start Eclipse SDK.

2. In the navigator view, expand the plugin project that you created.

    For more information on how to create a plugin project, see Section 6.4.3, "Creating a Plug-in Project".

3. To open the `plugin.xml` file in the Plugin Manifest Editor, do any of the following:

    • Double-click the `plugin.xml` file.

    • Right-click the `plugin.xml` file and select Open With > Plugin Manifest Editor.

4. Select the Extensions tab.

5. Click Add. From the New Extension window, select `org.eclipse.ui.actionSets` extension point from the list.

6. Click Finish.

7. Select the `org.eclipse.ui.actionSets` extension point. The Extension Details panel displays.

8. In the Extension Details panel, enter the properties of `org.eclipse.ui.actionSets` extension point.

9. Right-click on `org.eclipse.ui.actionSets` extension and select New > actionSet. The "actionSet" extension element gets created.

10. Select the "actionSet" extension element. The Extension Element Details panel displays.

11. In the Extension Details panel, enter the properties of the `actionSet` extension element.

12. In the visible, select "true" from the drop-down menu to make the actions defined in this action set visible.

13. To create a new `action` extension element, right-click on `actionSet` extension element and select New > action. The `action` extension element gets created.

14. Select the `action` extension element. The Extension Element Details panel displays.

15. In id*, enter a unique identifier for the action. This action identifier corresponds to the identifier of the action that gets invoked when you click on a menu item or related task. This action identifier also corresponds to the action identifier specified in the URL field, which is defined within a resource.

16. In class, to specify the implementation class, do any of the following:

> **Note:** This implementation class can extend either `YRCAction` class or `YRCRelatedTaskAction` class. If you are creating an normal action, which is used for menu items then extend the `YRCAction` class and if you want to create an related task action, which is used for related tasks then extend the `YRCRelatedTaskAction` class.

- Click Browse. The Select Type pop-up window displays. Select the class that extends the `YRCAction` class or `YRCRelatedTaskAction` class.

- Click on the class* hyperlink. The Java Attribute Editor window displays.
  - In Source Folder, the name of the source folder that you selected to store the action class automatically displays. You can also browse to the folder that you want to specify as the source folder.
  - In Package, the name of the package that you selected to store the action class automatically displays. This helps you to easily manage your directory structure.

- In Name, enter the name of the action class.

- In Superclass, click Browse, the Superclass Selection window displays.

- Select the YRCAction class or YRCRelatedTaskAction class from the list and click OK.

- Check the Constructors from superclass box. The system will automatically create the constructor for the superclass that you specified.

- Check the Inherited abstract methods box. The system will automatically add the abstract methods inherited by the superclass that you specified.

- Click Finish. The system will create the new action class in the folder or package selected by you.

17. Open the newly created action class in the Java Editor.

18. Depending on the action class that you are extending, write the code to perform the required operation in the inherited abstract execute() method. For example,

- If you are extending the YRCAction class then write the code for performing the required operation in the execute() method. The execute() method internally checks if the action can be executed or not. This check criteria depends on the following criteria:

  - Whether or not the current editor has errors.

  - Whether or not the current editor has been modified.

  Therefore, following methods must be overridden in the class which is extending the YRCAction class:

  - checkForErrors()—This method is used to check if the current editor has errors or not. If you want to skip this check, then return false.

  - checkForModifications()—This method is used to check if the current editor has been modified or not. If you want to skip this check, then return false.

- If you are extending theYRCRelatedTaskAction class then write the code for performing the required operation in the executeTask() method.

B.7 Registering a Plugin

Every plugin that is a part of the Sterling Rich Client application must be registered. Various features of Sterling Supply Chain Applications such as localization, theming, configuration, UI extension, and so forth depend on a plugin being registered. To register a plugin in the Sterling Rich Client application, you must invoke the registerPlugin() method of the YRCPlatformUI class.

> **Note:** When you create a Sterling RCP plugin using the Sterling RCP Wizards > UI wizards > Sterling RCP Plugin, the class for registering a plugin and other Sterling RCP-specific resource files is automatically created. Therefore, you need not explicitly register the plugin and other Sterling RCP-specific resource files. For more information about creating Sterling RCP plugin, see Section 6.4.4, "Running the Sterling RCP Plugin Wizard".

A sample code for registering a plugin is as follows:

```
public class TestPlugin extends AbstractUIPlugin {

    private static TestPlugin plugin;
    public static final String ID="com.mycompany.test.rcp";

    public TestPlugin() {
        super();
        plugin=this;
        try {
             YRCPlatformUI.registerPlugin(ID, this);
          } catch (Exception ex) {
               YRCPlatformUI.trace(ex);
          }
    }
}
```

B.8 Registering Plugin Files

Every plugin that wants to use its own resource files such as bundle, theme, configuration files, and so forth must register these files with Sterling Rich Client application. You can register all resource files together within the plugin constructor.

A sample code that can be used to register a plugin and all its resource files is as follows:

```
public class TestPlugin extends AbstractUIPlugin {

        private static TestPlugin plugin;
        public static final String ID="com.mycompany.test.rcp";

        public TestPlugin() {
                super();
                plugin=this;
                try {
YRCPlatformUI.registerPlugin(ID, this);
YRCPlatformUI.registerConfiguration("com.mycompany.test.rcp_
config", ID);
YRCPlatformUI.registerBundle("com.mycompany.test.rcp_bundle",
ID);
YRCPlatformUI.registerCommands("com.mycompany.test.rcp_
commands", ID);
YRCPlatformUI.registerExtensions("com.mycompany.test.rcp_
extn", ID);
YRCPlatformUI.registerTheme("com.mycompany.test.rcp_sapphire",
ID);
                } catch (Exception ex) {
                     YRCPlatformUI.trace(ex);
          }
}
```

B.8.1 Registering Bundle File

The bundle file is used for localizing Sterling Rich Client applications. Every plugin that requires its own bundle file should invoke the registerBundle() method of the YRCPlatformUI class during plugin initialization, preferably within the plugin constructor to register its bundle file. After the bundle file is registered, it gets loaded using the users current locale. The bundle file must have "properties" extension.

To register the bundle file within the plugin constructor, for example:

```
YRCPlatformUI.registerBundle("com.yantra.pca.ycd_bundle", ID)
```

where `com.yantra.pca.ycd_bundle` is the name of the bundle file without ".properties" extension. ID is a unique identifier of the plugin that registers this bundle file.

> **Note:** Before calling the registerBundle() method, the plugin must be registered using the registerPlugin() method of the YRCPlatformUI class. For more information on how to register a plugin, see Appendix B.7, "Registering a Plugin".

B.8.2 Registering Theme File

The theme file is used for setting the color scheme and font properties of Sterling Rich Client applications. Every plugin that requires its own theme file should invoke the registerTheme() method of the YRCPlatformUI class during plugin initialization, preferably within the plugin constructor to register its theme file.

To register the theme file within the plugin constructor, for example:

```
YRCPlatformUI.registerTheme("com.mycompany.test.rcp_skyblue",
ID)
```

where `com.mycompany.test.rcp_skyblue` is the name of the your theme file without ".ythm" extension. ID is a unique identifier of the plugin that registers this theme file.

> **Note:** Before calling the registerTheme() method, the plugin must be registered using the registerPlugin() method of the YRCPlatformUI class. For more information on how to register a plugin, see Appendix B.7, "Registering a Plugin".

B.8.3 Registering Configuration File

The configuration file is used to set the URL path parameters for connecting Sterling Rich Client applications to the server. Every plugin that requires its own configuration file should invoke the registerConfiguration() method of the YRCPlatformUI class during plugin initialization, preferably within the plugin constructor to register its configuration file. Configuration file must have extension "ycfg". Plugins can use any custom XML configuration file.

To register your configuration file within the plugin constructor, for example:

```
YRCPlatformUI.registerConfiguraton("com.mycompany.test.rcp_
config", ID)
```

where `com.mycompany.test.rcp_config` is the name of the your configuration file without ".ycfg" extension. ID is a unique identifier of the plugin that registers this configuration file.

> **Note:** Before calling the registerConfiguration() method, the plugin must be registered using the registerPlugin() method of the YRCPlatformUI class. For more information on how to register a plugin, see Appendix B.7, "Registering a Plugin".

B.8.4 Registering Commands File

The commands file is used to create commands to call different APIs or services. Every plugin that requires its own set of commands should invoke the registerCommands() method of the YRCPlatformUI class during plugin initialization, preferably within the plugin constructor to register its commands file. Commands file must have extension "ycml". The command names are unique, and reusing a command name overrides an existing definition. To register your commands file within the plugin constructor:

```
YRCPlatformUI.registerCommands("com.mycompany.test.rcp_
commands", ID)
```

where `com.mycompany.test.rcp_commands` is the name of the your commands file without ".ycml" extension. ID is a unique identifier of the plugin that registers this commands file.

> **Note:** Before calling the registerCommands() method, the plugin must be registered using the registerPlugin() method of the YRCPlatformUI class. For more information on how to register a plugin, see Appendix B.7, "Registering a Plugin".

B.8.5 Registering Extension File

The extension file is used to store information about Sterling Rich Client applications UI extensibility such as addition of new fields, modification of

existing fields, and so forth. Every plugin that requires its own extension file should invoke the registerExtensions() method of the YRCPlatformUI class during plugin initialization, preferably within the plugin constructor to register its extension file. The extension file must have "yuix" extension. To register your extension file within the plugin constructor:

```
YRCPlatformUI.registerExtensions("com.yantra.order.capture_
extn.yuix",ID)
```

where `com.yantra.order.capture_extn.yuix` is the name of the your extension file with ".extn" extension. ID is a unique identifier of the plugin that registers this extension file.

> **Note:** Before calling the registerExtensions() method, the plugin must be registered using the registerPlugin() method of the YRCPlatformUI class. For more information on how to register a plugin, see Appendix B.7, "Registering a Plugin".

B.9 Validating Controls

The Sterling RCP provides methods to validate various controls. When the controls have target binding, the associated data type is retrieved and appropriate validation is performed at the infrastructure level. You can validate the data entered in the controls such as text box, combo box, button, and so forth by implementing the appropriate validate method. The data type validation can be performed for the value entered by the user. Validations can also be performed for custom criteria. If the data type validation for a control fails, the validate method for that control is not called and an error message is displayed.

The methods for validating the following controls are:

- Text Control—When the text control loses focus, the data type validation and other mandatory validations are performed first. If the validation succeeds, the control is passed to the validateTextField() method.

- Combo Control—When a different item is selected from the combo control, the data type validation and other mandatory validations are performed first. If the validation succeeds, the control is passed to the validateComboField() method.

- Button Control—When the controls such as button, check box, radio button whether selected or unselected, the validateButtonClick() method is invoked.

B.10 Siblings

Siblings are the first level children of the parent. For example, let us consider the following scenario:
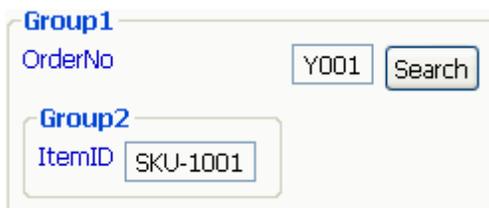
Here, the siblings of the OrderNo label are text box (Y001), button (Search), and Group2, which are the first level children of Group1. Similarly, the sibling of ItemID label is the text box (SKU-1001).

B.11 RCP Utilities

Sterling RCP provides a utility tool using which you can gather information for a particular UI or form such as form id, models used, and so forth.

B.11.1 Viewing Screen Models

Using the Screen Model utility tool you will know the form identifier and all models used in any UI, along with the various elements and attributes used in a model. You can also save all screen models as templates.

To view a screen model:

1. Run the appropriate Sterling Rich Client application.

2. After you successfully log in to the application, the application window displays.

3. Open the screen for which you want to view models.

4. Press CTRL+SHIFT+M to view screen model. The Screen Models window displays as shown in Figure .
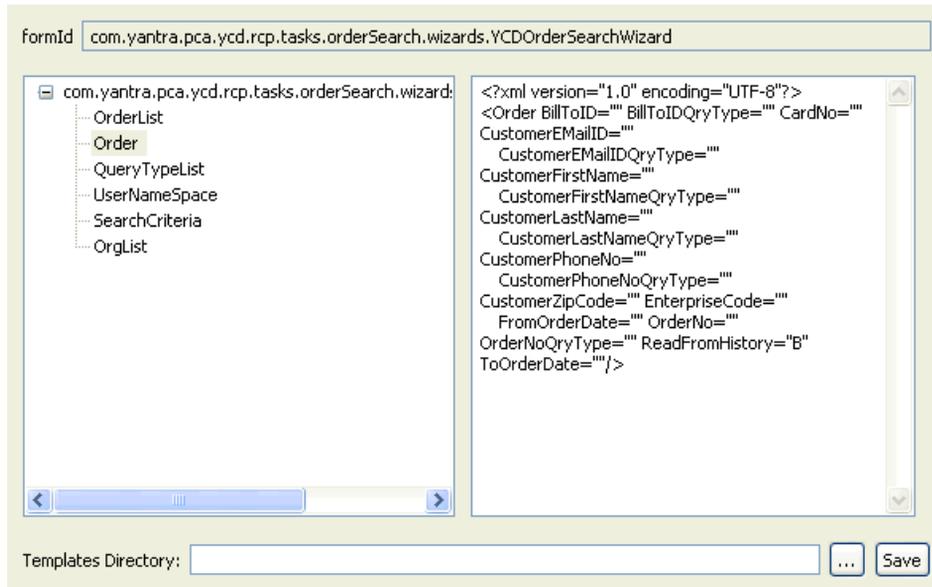
In the formId field, the identifier of the form displays, which is used to identify the screen.

The left-hand side panel displays a list of models used in the screen as a tree structure with root being the form identifier of the screen. After you select a specific model, you can view a list of elements and attributes defined in the model on the right-hand side panel. If a screen contains embedded screens in it, then you can view a list of models used for each screen.

B.11.1.1 Saving Models as Templates
To save existing models as templates:

1. Click the button next to the Templates Directory field. The Choose Directory pop-up window displays.

2. Select the directory where you want to store the models of the screen as templates, and click OK.

3. Click Save. The system stores the models of each screen as templates in their respective folders.

C

Console JSP Interface Extensibility Reference

Customizing the user interface requires writing scripts that determine how the user interface renders the screen and passes data. This chapter contains the supporting reference material necessary for customizing the user interface, as described in Chapter 3, "Customizing the Console JSP Interface".

> **Note:**   The HSDE screens are not extensible.

This chapter contains the following resources for your reference while developing the user interface:

- User Interface Style Reference
- Programming Standards
- JSP Functions
- JSP Tag Library
- JavaScript Functions
- Data Type Reference

Also, as when doing any sort of programming, see Appendix A, "Special Characters Reference".

C.1 User Interface Style Reference

The style reference helps you maintain consistency and enables easy code reusability.

C.1.1 Controls and Classes

This section is a quick reference list of the most common types of HTML controls and their corresponding CSS class tags. The typical controls and the corresponding CSS classes used in a JSP file are listed in Table C–1.

Table C–1   Typical Controls and Tags

Control and Tag	Available Classes	Description
Buttons	button	For all buttons.
Checkboxes	checkboxcolumn	For checkboxes displayed in a column of a table.
	checkboxheader	For checkboxes displayed in the header of a table. Use this class at the <td> cell level, not the <input> tag level.
Comboboxes/ Selects	combobox	For all combo boxes.
Icons	columnicon	For icons in a table column.
	icon	For icons not in a table column.
	lookupicon	For lookup icons appearing to the right of certain editable input texts.
Input Boxes	dateinput	For editable date input.
	numericprotectedinput	For non-editable numeric input. Right-aligned. Surrounding <td> tag should have class="protectednumber".
	numericunprotectedinput	For editable numeric input. Right-aligned.
	protectedinput	For non-editable input. Surrounding <td> tag should have class="protectedtext".
	unprotectedinput	For editable input.

Table C–1   Typical Controls and Tags

Control and Tag	Available Classes	Description
Labels	N/A	Takes the same font as specified by the table class in which it resides. Use detail labels for a detail view and search labels for a search view.
Radio Buttons	radiobutton	For all radio buttons.
Tables (non-tabular data)	view	For detail views. This class is attached to `view.htc`, which dynamically sets column widths when the HTML page loads.
Tables (tabular data)	table	For all tables of the tabular data type. Specify the following attributes for the table element: • editable="true" • deleteAllowed="true" • addAllowed="true" The heading row must be included in a <thead> tag. Heading cells must be in <td> tag and not the <th> tag. The body rows must be included in a <tbody> tag. If the table is an editable list that permits add, specify a template row in a <tfoot> tag. Specify the following attributes for the template row <tr> tag: • "style="display:none" • TemplateRow="true" • ByPassRowColoring="true"
Text	numerictablecolumn	For displaying text in a table column. Right-aligned.
	protectednumber	For displaying numeric data. Right-aligned.
	protectedtext	For displaying text.

Table C–1   Typical Controls and Tags

Control and Tag	Available Classes	Description
	searchcriteriacell	For the bottom <td> tag of each search criteria. **Note**: all <td> tags using class as searchcriteriacell must specify nowrap="true". This prevents the lookup icon from wrapping to the next line when used with text boxes, input boxes, and query combo boxes.
	tablecolumn	For displaying non-editable text in a table column.
	tablecolumnheader	For displaying text in a table column header.
Textarea	textarea	For keeping text areas consistent. See also getTextAreaOptions().
Total fields	totaltext	Shows text in a different color to identify a total field.

C.1.2 Page Layout

The following table describes the guidelines to follow when customizing screens in the Sterling Supply Chain Application Consoles.

Table C–2   Page Layout Style Guidelines

Object	Standards
Anchor Page	Defines the layout of the inner panels to be included on a screen. For example, the Order Detail anchor page includes all inner panels relevant to Order Detail and defines how they should be laid out. Each entity should have its own anchor page.
	Use the following standards when developing anchor pages:
	Table tags - The outermost, or container, <table> tag should contain `cellspacing="0"` and `cellpadding="0"` attributes as the spacing is achieved through the classes used for the inner panels themselves. If these attributes are not set to "0" the amount of spacing and padding could potentially be inconsistent.
	Cell tags - <td> tags within the same <tr> tag should contain `height="100%"` to ensure the horizontally aligned cells are the same height.
Inner Panel Title-bars	Access to available popup windows granted through left-aligned icons in the title-bar.
	Available actions included in the right-aligned drop-down in the title-bar.
	Both of the above are achieved through the Sterling Service Definition Framework.
Insets	Should be laid out symmetrically and consistent with other screens.
	Should be spaced five pixels from each other and from the edge of the main page. Apply this spacing to the inner panels, starting with the top inner panel and working down and to the right. For each individual inner panel, apply spacing to the top, left side, right side and then the bottom. Be careful not to add more spacing than necessary. For example, if there are two inner panels horizontally aligned and the left panel has been specific a right spacing of 5px, the right panel does not require left spacing of 5px, as this would result in total spacing of 10px.

Table C–2   Page Layout Style Guidelines

Object	Standards
Labels and Inputs	Specific names for labels on the screen should be the same as the names used in the Sterling Supply Chain Applications Configurator. If not used in the Sterling Supply Chain Applications Configurator, the labels should be the same as console in previous releases.
	Should be vertically spaced 5 pixels from one another. There is no default value for padding or spacing.
List Checkboxes	Checkboxes that appear in lists to enable the user to select specific rows. These should appear to the left of each row and is coded within the JSP that displays them.
Lists	All columns displaying numeric values should be right aligned.
Menu Bar	Should be displayed at the top of all pages that are not popup windows.
Page Title-bars	Should describe the current page and contain a list of available views when there is more than one view available.
Search Criteria	All available search criteria text inputs should be preceded by a combobox. For text search fields, provide combobox options for *is*, *starts with*, and *contains*. For numeric search fields, provide combobox options for *less than*, *greater than*, and *equal to*.
	Lookups should be provided to the right of the text inputs.

C.1.3 Hypertext Links

When a screen has a field that is a logical reference to another entity, hyperlink the data of that field to the entity to which it refers. For example, hyperlink the Order# field on the Order Line Detail screen to the Order screen.

C.2 Programming Standards

Follow these programming standards to help you create and maintain files that are consistent, easy-to-read, and reusable.

C.2.1 Standards for Creating Well-Formed JSP Files

Although HTML code is embedded in Java Server Pages, strive to write JSP code that is easily readable. If you require some special XML manipulation that cannot be incorporated in the APIs, include a separate JSP file, so that HTML tags and Java code do not become mixed together.

Use the following standards when writing JSP files:

- Tab spacing - Set the editor tab spacing to 4.

- JavaScript files - Do not include any JavaScript in the JSP file. Put all JavaScript into a separate JS file.

- HTML tags - Type all HTML tags and attributes in lowercase letters.

- HTML attributes - Enclose all HTML element attribute values in double quotes. Single quotes and no quotes may work, but the standard is to use double quotes.

- HTML tables - Minimize the number of tables in HTML pages. Especially, reduce the number of nested tables (a table within another table).

- Tags - Close all tags, whether required or not.

- Control elements - For each control element, add the `get…Options` attribute as the last attribute for that control element.

- Comments - Enclose all comments in the following manner:
  `<%/*……..*/%>`

> **Tip:** When finished coding a form, open it in any visual HTML editor to validate that the HTML is well-formed.

C.2.2 Valid HTML Tags and Attributes

Follow this HTML reference material to help guide you in using the HTML attributes as they are used by Sterling Supply Chain Applications.

> **Note:** You can also use any other HTML attributes, as long as you devise your own set of standards.

Table C–3 lists the recommended standard HTML tags and their attributes. For each HTML tag, use only the attributes listed in the Tag Attribute column.

Table C–3   Recommended Standard HTML Tags and Attributes

HTML Tag	Tag Attributes
<% %>	keep at the top of the JSP wherever possible
<a>	href
<imp>	alt
	border
	name
	src
	style
<input>	class
	maxlength
	name
	onblur
	style
	value
<option>	binding
	selected
	type
	value
<select>	class
	name
<table>	editable

Table C–3   Recommended Standard HTML Tags and Attributes

HTML Tag	Tag Attributes
	cellPadding
	cellspacing
	class
	style
<tbody>	N/A
<td>	class
	colspan
	nowrap
	onclick
	rowspan
	sortable
	sortValue
	style
<tfoot>	N/A
<thead>	N/A
<tr>	style
	templateRow (true/false)

C.2.3 Conventions for Naming JSP Files and Directories

As you populate directories with JSP files, adhere to a consistent hierarchical directory structure and a consistent file naming convention.

Use the following rules when choosing names for JSP files:

- Do not use any capital letters.

- Use underscores, not hyphens, to separate words.

- If the JSP file is an anchor page, include the word *anchor* in the name.

Directory and File Name Syntax

<module>/<entity>_<screen_type>_<viewdesc|"anchor">.jsp

Example

`om/orderline/search/orderline_search_bydate.jsp`

<**module**> represents a two-character module code. For example:

- cm - Catalog Management
- em - Alert Management
- im - Inventory Management
- om - Order Management
- pm - Participant Management

<**entity**> represents the resource ID of the entity. For example:

- order - Order entity
- orderline - Order Line entity
- orderrelease - Order Release entity

<**screen_type**> represents the resource type of the view. For example:

- list - List views
- detail - Detail views
- search - Search views

<**viewdesc**> represents an abbreviated description of the view or the inner panel. For example:

- primaryinfo - Primary Information
- paymentinfo - Payment Information
- collectiondtl - Collection Details

C.2.4 Conventions for Naming Controls

When using the Sterling Presentation Framework function for XML binding the input controls in your JSP, you should not set the `name` attribute for that control. In general, avoid naming each control. Instead, try to access the control through the HTML object or DOM hierarchy. If

you do need to name a control, keep in mind that the name must be unique within each page.

C.2.5 Internationalization

Sterling Presentation Framework enables you to write an internationalized application by providing the following features that can be customized to be *locale-specific*:

- i18n JSP tag for literals

- Graphics and images

- Client-side error messages

- Date and number validations

- Themes (which includes fonts that support language character sets)

C.2.6 Validating Your HTML and CCS Files

Validate both your HTML and CSS files. You can use any commercial software package or free, online application, such as the following World Wide Web Consortium (W3C) validators:

- W3C CSS Validator at http://jigsaw.w3.org/css-validator/

- W3C HTML Validator at http://validator.w3.org/

As an alterative, when you finish coding a form, you can open it in any visual HTML editor to validate that the HTML is well-formed.

C.3 CSS Theme File Reference

This section contains reference material to help guide you when modifying the CCS files.

The standard Sterling Supply Chain Applications theme uses the Tahoma font as specified within the CSS files. If you need to use a different sized font, you may encounter display problems, such as truncation of the drop-down list items. In this event, you can edit the CSS file and specify properties that will enable the screen to display correctly. Use the classes and properties described in Table C–4.

Table C–4   CSS Theme Classes

Class	Description
favouritespopuprowhighlight	Drop-down list items under the Favorite folder icon to highlight during mouse over actions. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the favouritespopuprownormal class. • charwidth - horizontal size of character. Use the same value as specified for the favouritespopuprownormal class.
favouritespopuprownormal	Drop-down lists under the Favorite folder icon. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the favouritespopuprowhighlight class. • charwidth - horizontal size of character. Use the same value as specified for the favouritespopuprowhighlight class.
ipactionspopuprowhighlight	Drop-down list items on detail views to highlight during mouse over actions. Contains both header and line information. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the ipactionspopuprownormal class. • charwidth - horizontal size of character. Use the same value as specified for the ipactionspopuprownormal class.

Table C–4   CSS Theme Classes

Class	Description
ipactionspopuprownormal	Drop-down list items on detail views. Contains both header and line information. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the ipactionspopuprowhighlight class. • charwidth - horizontal size of character. Use the same value as specified for the ipactionspopuprowhighlight class.
listactionspopuphighlight	Drop-down list items on list views to highlight during mouse over actions. On list views. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the listactionspopupnormal class. • charwidth - horizontal size of character. Use the same value as specified for the listactionspopupnormal class.
listactionspopupnormal	Drop-down list items on list views. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the listactionspopuphighlight class. • charwidth - horizontal size of character. Use the same value as specified for the listactionspopuphighlight class.

Table C–4   CSS Theme Classes

Class	Description
menuitempopuprowhighlight	Drop-down list items on the menu bar to highlight during mouse over actions. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the menuitempopuprownormal class. • charwidth - horizontal size of character. Use the same value as specified for the menuitempopuprownormal class.
menuitempopuprownormal	Drop-down list items on the menu bar. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the menuitempopuprowhighlight class. • charwidth - horizontal size of character. Use the same value as specified for the menuitempopuprowhighlight class.
menulevel1hl	Menu bar items to highlight during mouse over actions. Has the following properties: • height - background vertical size.
menulevel1norm	Menu bar items. For example Order, Supply, System Management, and so forth: Has the following properties: • height - background vertical size.

Table C–4   CSS Theme Classes

Class	Description
searchentitiespopuprowhighlight	Drop-down list items (left side) on search views to highlight during mouse over actions. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the searchentitiespopuprownormal class. • charwidth - horizontal size of character. Use the same value as specified for the searchentitiespopuprownormal class.
searchentitiespopuprownormal	Drop-down list items (left side) on search views. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the searchentitiespopuprowhighlight class. • charwidth - horizontal size of character. Use the same value as specified for the searchentitiespopuprowhighlight class.

Table C—4   CSS Theme Classes

Class	Description
searchviewspopuprowhighlight	Drop-down list items (right side) on search views to highlight during mouse over actions. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the searchviewspopuprownormal class. • charwidth - horizontal size of character. Use the same value as specified for the searchviewspopuprownormal class.
searchviewspopuprownormal	Drop-down list items (right side) on search views. Has the following properties: • charheight - vertical size of character. Use the same value as specified for the searchviewspopuprowhighlight class. • charwidth - horizontal size of character. Use the same value as specified for the searchviewspopuprowhighlight class.

C.4 JSP Functions

This section describes the JSP functions that you can use.

C.4.1 changeSessionLocale

While locale is configured at the user level, you can also dynamically switch to a specific locale by using this JSP function.

Syntax
void changeSessionLocale(String localecode)

Input Parameters

localecode - Locale you want to switch to.

C.4.2 equals

This JSP function is a cover over Java's equal function that takes care of the situation when either of the objects are null or contain zero or more white spaces. In all of those situations, the two objects are considered equal.

Syntax

boolean equals(Object obj1, Object obj2)

Input Parameters

obj1, obj2 - Required. The two objects that must be compared.

Example

This example shows how this function makes string comparisons.

```
<% String sAvailable="";
if(equals(resolveValue("xml:/InventoryInformation/Item/@TrackedEverywhere"),"N")
)
        sAvailable=getI18N("Available") + ": " + getI18N("INFINITE");
    else
        sAvailable=getI18N("Available") + ": "
+resolveValue("xml:/InventoryInformation/Item/@AvailableToSell");
%>
```

C.4.3 getCheckBoxOptions

This JSP function is a standard function to XML bind checkboxes when modification rules do not need to be considered.

Syntax

String getCheckBoxOptions(String name)

String getCheckBoxOptions(String name,String a_checked, String a_ value)

Input Parameters

name - Required. Value of the name attribute for the checkbox input. Can be a binding or a literal.

checked - Required. When the value of the value attribute is equal to this value, the CHECKED attribute is set to true.

value - Required. Value of the value attribute for the checkbox input. Can be a binding or a literal.

> **Note:** If only the name parameter is passed, value defaults to the same value passed to the name parameter.

JSP Usage

```
<input type="checkbox"
<%=getCheckBoxOptions("xml:Order:/Order/Addlinfo/@Country" ,"IND",
"xml:Order:/Order/Addlinfo/@Country" )%></yfc:i18n>India</yfc:i18n></input>
```

Resultant HTML

```
<input type="checkbox" name="xml:Order:/OrderAddlinfo/@Country"
value="IND">India</input>
```

C.4.4 getColor

This JSP function returns the HTML color in hexidecimal code based on the specific color object.

Syntax

public String getColor(java.awt.Color color)

Input Parameters

color - Required. Color object.

C.4.5 getComboOptions

This JSP function provides a standard function to XML bind combo boxes when modification rules do not need to be considered.

Syntax

String getComboOptions(String name)

String getComboOptions(String name, String value)

Input Parameters

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. Through the Sterling Presentation Framework, the target XML is then passed to the appropriate API.

value - Required. Specifies the value to be selected in the combobox. Can be a binding or a literal.

JSP Usage

This example shows how to render the enterprise code combobox in the Order Entry screen. The API is used in conjunction with the loopOptions JSP tag.

```
            <select class="combobox" onChange="updateCurrentView()"
<%=getComboOptions("xml:/Order/@EnterpriseCode",enterpriseCode)%>>
                <yfc:loopOptions binding="xml:/OrganizationList/@Organization"
name="OrganizationCode"
                value="OrganizationCode" selected="xml:/Order/@EnterpriseCode"/>
            </select>
```

C.4.6 getComboText

This JSP function provides a standard function to get description from a list of values.

Syntax

String getComboText(String binding, String name, String value, String selected)

String getComboText(String binding, String name, String value, String selected,boolean localized)

Input Parameters

binding - Required. Binding string that points to the repeating element in the API output. The repeating element must be one fixed with the at character ("@").

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. Through the Sterling Presentation Framework, the target XML is then passed to the appropriate API.

value - Required. Specifies the value to be selected in the combobox. Can be a binding or a literal.

selected - Optional. Binding string that must be evaluated and set as the default selected value. This is matched with the value attribute, not the description attribute. Defaults to blanks, for example, space (" ").

localized - Optional. If passed as true, it fetches the localized description to be displayed.

JSP Usage

This example shows how to render the enterprise code combobox in the Order Entry screen. The API is used in conjunction with the `loopOptions` JSP tag.

```
<%=getComboText("xml:TaxNameList:/CommonCodeList/@CommonCode","CodeShortDescription","CodeValue","xml:/HeaderTax/@TaxName",true)%>
```

C.4.7 getDateOrTimePart

This JSP function returns a string representing the date or time portion of a timestamp value.

Syntax

String getDateOrTimePart(String type, String value);

Input Parameters

type - Required. Specifies whether to display the date or time. Pass YFCDATE to return the date portion of the timestamp. Pass YFCTIME to return the time portion of the timestamp.

value - Required. String containing a timestamp value in the current login user's locale's timestamp format.

Output Parameters

A string representing the date or time portion of the timestamp attribute.

Example

This example uses the `getDateOrTimePart()` function to return the date portion of the attribute referred to in the `xml:/OrderRelease/@HasDerivedParent` binding.

```
getDateOrTimePart("YFCDATE",
resolveValue("xml:DeliveryPlan:/DeliveryPlan/@DeliveryPlanDate));
```

C.4.8 getDateValue

This JSP function retrieves the date from an XML in XML format, and not in the format of current locale. This is typically used for storing a custom `sortValue` attribute in a column for sorting a table.

Syntax

String getDateValue(String bindingStr);

Input Parameters

bindingStr - Required. Binding string that must be resolved into a date string.

Output Parameters

Date string (YYYYMMDDHH24MISS structure), with the following values:

YYYY - Required. Four-digit display of year (for example, 2002).

MM - Required. Two-digit display of month (for example, 05 for May).

DD - Required. Two-digit display of date (for example, 05 for 5th).

HH24 - Required. Two-digit display of hour on a 24-hour scale (for example, 16 for 4 PM).

MI - Required. Two-digit display of minutes.

SS - Required. Two-digit display of seconds.

Example

This example shows how the `getDateValue()` function stores the date in this format for subsequent client-side sorting by the user on the list of alerts for an order.

```
<table class="table" editable="false" width="100%" cellspacing="0">
    <thead>
```

```
            <tr>
                <td class="tablecolumnheader"><yfc:i18n>Alert_ID</yfc:i18n></td>
                <td class="tablecolumnheader"><yfc:i18n>Raised_On</yfc:i18n></td>
            </tr>
        </thead>
        <tbody>
            <yfc:loopXML binding="xml:/InboxList/@Inbox" id="Inbox">
                <tr>
                    <td class="tablecolumn">
                            <yfc:getXMLValue binding="xml:/Inbox/@InboxKey"/>
                    </td>
                    <td class="tablecolumn"
    sortValue="<%=getDateValue("xml:Inbox:/Inbox/@GeneratedOn")%>">
    <yfc:getXMLValue binding="xml:/Inbox/@GeneratedOn"/>
    </td>
                </tr>
            </yfc:loopXML>
        </tbody>
    </table>
```

C.4.9 getDBString

This JSP function returns a string representing the localized version of
the input string. The input to this method should be a data string that
has been translated in the YFS_LOCALIZED_STRINGS table.

Syntax

String getDBString(String inString)

Input Parameters

inString - Required. The string to be translated. This string should be
translated in the YFS_LOCALIZED_STRINGS table.

Example

This example shows how to display the translated version of the
Description attribute of the singleDocType element.

```
<%=getDBString(singleDocType.getAttribute("Description"))%>
```

C.4.10 getDetailHrefOptions

This JSP function is typically used to form a link in an inner panel that opens another detail view. A link is modeled as a resource of type Link. The resource can point to any other detail view, and you can configure this through the resource configurator. Use this function inside an <a> tag. This function can be used only in an inner panel (and therefore only in a detail view).

Syntax

String getDetailHrefOptions(String linkIdSuffix, String entityKey, String extraParams)

Input Parameters

linkIdSuffix - Required. Link ID suffix. A resource of type Link is named as <current inner panel's Resource ID><suffix>. For instance, if the current inner panel's Resource ID is YOMD010I01, a link ID is YOMD010I01L01 and the suffix is `L01`. Pass only the suffix `L01`.

entityKey - Required. Key (formed through the `makeXMLInput` JSP tag) that must be passed on to the view that is invoked by selecting this link.

extraParams - Required. String containing extra parameters that are appended to the URL that is formed for the <a href> tag. The String should start with an ampersand ("&") and should contain name-value pairs in the `<name>=<value>` format. Restrict the use of this parameter only to cases where it is absolutely necessary because there is a size limit of what can be passed in a URL. Typically, each view should only take the key and retrieve other details from an API based on that key.

Output Parameters

A string containing `href=""` and `onclick=""` attributes must be plugged into a <a> tag in HTML.

The resource of type link is not permission controlled. However, the view to which a link points is permission controlled. Still, since this function is called inside an <a> tag, the link is formed regardless of whether or not the user has permissions for the view to which the link points. If the user selects the link, the view that is displayed gives an Access Denied message.

Example

This example shows how the `getDetailHrefOptions()` function forms a hyperlink to the Alert Detail view from a list of alerts for an order.

```
<table class="table" editable="false" width="100%" cellspacing="0">
    <thead>
        <tr>
            <td sortable="no" class="checkboxheader">
                <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
            </td>
            <td class="tablecolumnheader"><yfc:i18n>Alert_ID</yfc:i18n></td>
        </tr>
    </thead>
    <tbody>
        <yfc:loopXML binding="xml:/InboxList/@Inbox" id="Inbox">
            <tr>
                <yfc:makeXMLInput name="inboxKey">
                    <yfc:makeXMLKey binding="xml:/Inbox/@InboxKey"
value="xml:/Inbox/@InboxKey"/>
                </yfc:makeXMLInput>
                <td>
                    <input type="checkbox" value='<%=getParameter("inboxKey")%>'
name="EntityKey"/>
                </td>
                <td class="tablecolumn">
                    <a <%=getDetailHrefOptions("L01",
getParameter("inboxKey"),"")%> >
                        <yfc:getXMLValue binding="xml:/Inbox/@InboxKey"/>
                    </a>
                </td>
            </tr>
        </yfc:loopXML>
    </tbody>
</table>
```

C.4.11 getDetailHrefOptions (with additional parameter)

This JSP function is similar to the getDetailHrefOptions() function, except that it takes an additional parameter. This additional parameter enables you conditionally link to different views with the same hyperlink. First, configure multiple link resources under the same inner panel that have the same link ID except for a conditional suffix. For example, configure one link with ID YOMD010I01L010001 that points to one view and

another link with ID YOMD010I01L010002 that points to a different view. Then, in the JSP, you can use this function within an <a> tag to conditionally link to different views by passing different values for the conditionalLinkId parameter.

Syntax

getDetailHrefOptions(String linkIdSuffix, String conditionalLinkId, String entityKey, String extraParams)

Input Parameters

linkIdSuffix - Required. Link ID suffix. A resource of type Link is named as <current inner panel's Resource ID><suffix>. For instance, if the current inner panel's Resource ID is YOMD010I01, a link ID is YOMD010I01L01 and the suffix is `L01`. Pass only the suffix `L01`.

conditionalLinkId - Required. Portion of the suffix of the link ID. Used to conditionally link to different views.

entityKey - Required. Key (formed through the `makeXMLInput` JSP tag) that must be passed on to the view that is invoked by selecting this link.

extraParams - Required. String containing extra parameters that are appended to the URL that is formed for the <a href> tag. The string should start with an ampersand (&) and contain name value pairs in the syntax <name>=<value>. Because there is a size limit on what can be passed in a URL, use this parameter only when absolutely necessary. Typically, each view should only take the key and retrieve other details from an API based on that key.

Output Parameters

A string containing `href=""` and `onclick=""` attributes that must be plugged into a <a> tag in HTML.

The resource of type link is not permission controlled. However, the view to which a link points is permission controlled. Still since this function is called inside an <a> tag, the link is formed regardless of whether or not the user has permissions for the view to which the link points. If the user selects the link, the view that is displayed gives an Access Denied message.

Example

This function is useful when a specific hyperlink on a screen must link across document types. For example, a list of shipments on a Delivery Plan screen could be shipments for different document types (order and purchase order shipments). The detail view that must be shown for the two types of shipments is different. The document type of the shipment can be used as the conditionalLinkId.

```
<a <%=getDetailHrefOptions("L01", getValue("Shipment",
"xml:/Shipment/@DocumentType"), getParameter("shipmentKey"), "")%>>
<yfc:getXMLValue binding="xml:/Shipment/@ShipmentNo"/>
</a>
```

This example shows the call to the getValue() function returns the document type of the shipment that is used as the conditionalLinkId. For this example to work, the inner panel using this JSP should have to link resources defined with the following properties:

```
Link 1: ID="YDMD100I02L010001" View ID="YOMD330"
Link 2: ID="YDMD100I02L010005" View ID="YOMD7330"
getDoubleFromLocalizedString -
```

C.4.12 getDoubleFromLocalizedString

This JSP function returns a double value that is represented in a string containing the number in the format used by a particular locale.

This function does the reverse of the getLocalizedStringFromDouble() function. See "getLocalizedStringFromDouble" on page 612.

Syntax

double getDoubleFromLocalizedString(YFCLocale aLocale, String sVal)

Input Parameters

aLocale - Required. The YFCLocale object for which you want the number formatted for a specific locale.

sVal - Required. The string containing the formatted representation of the number.

Output Parameters

A double containing the unformatted number.

JSP Usage

This example shows how the string variable containing a formatted double called `sTotalInternalUnassignedDemand` is compared to zero by first converting it into the double value.

```
<% if ((getDoubleFromLocalizedString(getLocale(),
sTotalInternalUnassignedDemand)) > 0) {%>
   <table  border="1" style="border-color:Black" cellspacing="2" cellpadding="2"
   bgcolor="<yfc:getXMLValue
   binding="xml:/InventoryInformation/Item/InventoryTotals/Demands/@Total
   InternalUnassignedDemand" />">
       <tr>
     <td style="height:10px;width:15px"></td>
       </tr>
   </table>
<%}%>
```

C.4.13 getElement

This JSP function gets the YFCElement object that resides in a specific namespace. You can use this function to obtain a handle to the YFCElement and subsequently manipulate the XML in the YFCElement object.

YFCElement is part of Sterling Supply Chain Applications DOM utility package. To see the APIs available in this package, see the *Sterling Supply Chain Applications Javadocs*.

Syntax

YFCElement getElement(String nameSpace)

Input Parameters

nameSpace - Required. Namespace that contains the YFCElement needs to be returned.

Output Parameters

YFCElement - Required. YFCElement object that resides in the namespace provided.

Example

This example shows how the Return detail view controls whether the active or inactive state of the Schedule operation uses this function.

The Schedule operation is not valid for draft orders.

The `getOrderDetail()` API returns DraftOrderFlag attribute in the XML.

This flag is Y when the order is draft order, and N otherwise.

This must be converted into another flag that is opposite in meaning. As a result, use an attribute called ConfirmedFlag, which is N when the order is a draft order and Y when the order is no longer a draft order.

```
<%
    YFCElement elem=getElement("Order");
    if (elem != null) {
        //Flip the draft order flag into confirmed flag.
        elem.setAttribute("ConfirmedFlag",
!isTrue("xml:/Order/@DraftOrderFlag"));
    }
%>
```

C.4.14 getImageOptions

This is the JSP function used for building an image tag in HTML.

A Java constants file keeps the image path and icon centralized. If the path starts with `/yantra/console/icons`, the image file is first searched for inside `/webpages/extn/icons/yantraiconsbe.jar` (or the localized icons JAR file) and then inside the `/webpages/yfscommon/yantraiconsbe.jar` (or the localized icons JAR file). The path to be specified is the path of the image file inside the JAR file.

If the path does not start with `/yantra/console/icons`, it retrieves the file from the location specified in the EAR file. It is strongly advised that you place your images under the `/console/icons/` directory in the custom icons JAR file (`yantraiconsbe.jar`).

The path to be specified is the path of the image file inside the JAR file.

If you want to use an image that may be hidden based on modification rule considerations, use the yfsgetImageOptions() function. See "yfsGetImageOptions" on page 630.

Syntax
getImageOptions(imgfilewithpath, alt)

Input Parameters
imgfilewithpath - Required. Full path to the file of the image.

alt - Required. String to use as the alt attribute for the image. This string is displayed when the image cannot be rendered on screen.

JSP Usage
```
<img class="lookupicon" name="search"
<%=getImageOptions("/yantra/console/icons/shipnode.gif" "Search_for_
Organization") %> />
```

C.4.15 getLocale
This JSP function returns a YFCLocale object that represents the locale of the user logged into Sterling Supply Chain Applications.

Syntax
YFCLocale getLocale()

Input Parameters
None.

Output Parameters
The YFCLocale object that represents the locale of the logged in user.

JSP Usage
This example shows how the getLocale function can be used in conjunction with the getDoubleFromLocalizedString function. See "getDoubleFromLocalizedString" on page 608.

```
<% if ((getDoubleFromLocalizedString(getLocale(),
sTotalInternalUnassignedDemand)) > 0) {%>
   <table  border="1" style="border-color:Black" cellspacing="2" cellpadding="2"
   bgcolor="<yfc:getXMLValue
   binding="xml:/InventoryInformation/Item/InventoryTotals/Demands/@TotalInter
   nalUnassignedDemand" />">
      <tr>
```

```
            <td style="height:10px;width:15px"></td>
        </tr>
    </table>
<%}%>
```

C.4.16 getLocalizedStringFromDouble

This JSP function returns a representation of a string value and displays it in the correct format for a specific locale.

Sterling Supply Chain Applications always display numeric data in the format specific to the locale of the logged in user. If you have a decimal value that you need to display to the user that is not formatted in any locale, use this function to get a string representing the correctly formatted representation of the decimal value.

This function does the reverse of the getDoubleFromLocalizedString() function. See "getDoubleFromLocalizedString" on page 608.

Syntax

String getLocalizedStringFromDouble(YFCLocale aLocale, double aDblVal)

Input Parameters

aLocale - Required. The YFCLocale object for which you want the number formatted for a specific locale.

aDblVal - Required. The number (which can include decimals) you want formatted for a specific locale.

Output Parameters

A string containing the correctly formatted representation of the number.

JSP Usage

This example shows how to get the localized format of the number 2500.75. If the locale used is en_US, then the sBalance variable is 25,00.75.

```
String sBalance = getLocalizedStringFromDouble(locale, 2500.75);
```

C.4.17 getLocalizedStringFromInt

This JSP function returns a representation of an integer value and displays it in the correct format for a specific locale.

Sterling Supply Chain Applications always display numeric data in the format specific to the locale of the logged in user. If you have a decimal value that you need to display to the user that is not formatted in any locale, use this function to get a string representing the correctly formatted representation of the integer value.

Syntax
String getLocalizedStringFromInt(YFCLocale aLocale, int intVal)

Input Parameters
aLocale - Required. The YFCLocale object for which you want the number formatted for a specific locale.

intVal - Required. The integer you want formatted for a specific locale.

Output Parameters
A string containing the correctly formatted representation of the number.

JSP Usage
This example shows to display an integer variable called `quantity` within a <td> tag.

```
<td class="protectednumber">
   <%=getLocalizedStringFromInt(getLocale(), quantity)%>
</td>
```

C.4.18 getLoopingElementList

This JSP function can be used as an alternative to the loopXML JSP tag. See "loopXML" on page 643.

If your application servers only supports up to JSP specification version 1.1, and you need to include another JSP (using `jsp:include`) within the loop, use this function.

Syntax
ArrayList getLoopingElementList(String binding)

Input Parameters

binding - Required. The XML binding to the element within an XML that you want to repeat.

Output Parameters

An ArrayList containing the list of elements that you can then use in a loop.

JSP Usage

This example loops on the `xml:PromiseList:/Promise/Options/@Option` element. For each iteration of the loop, it includes the `/om/lineschedule/list/lineschedule_list_option.jsp` JSP file.

Note that loop element is set into an attribute of the pageContext so that it will be available within the included JSP.

```
<td colspan="6" style="border:1px ridge black">
    <% ArrayList optList =
getLoopingElementList("xml:PromiseList:/Promise/Options/@Option");
for (int OptionCounter = 0; OptionCounter < optList.size(); OptionCounter++) {

   YFCElement singleOpt = (YFCElement) optList.get(OptionCounter);
   pageContext.setAttribute("Option", singleOpt); %>

<% request.setAttribute("Option",
(YFCElement)pageContext.getAttribute("Option")); %>
<jsp:include page="/om/lineschedule/list/lineschedule_list_option.jsp"
flush="true">
</jsp:include>
    <% } %>
</td>
```

C.4.19 getNumericValue

This JSP function retrieves a number from an XML output in the original XML format, and not in the format of current locale. This is typically used for storing a custom `sortValue` attribute in a column for sorting a table.

Syntax

String getNumericValue(String bindingStr)

Input Parameters

bindingStr - Required. Binding string that must be resolved into a number string.

Output Parameters

Number string in Sterling Supply Chain Applications XML format, with decimals.

Example

This example shows how the `getNumericValue()` function is used to store the priority in XML format for subsequent client-side sorting by user on the list of alerts for an order.

```
<table class="table" editable="false" width="100%" cellspacing="0">
    <thead>
        <tr>
            <td sortable="no" class="checkboxheader">
                <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
            </td>
            <td class="tablecolumnheader"><yfc:i18n>Alert_ID</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Priority</yfc:i18n></td>
        </tr>
    </thead>
    <tbody>
        <yfc:loopXML binding="xml:/InboxList/@Inbox" id="Inbox">
            <tr>
                <yfc:makeXMLInput name="inboxKey">
                    <yfc:makeXMLKey binding="xml:/Inbox/@InboxKey"
value="xml:/Inbox/@InboxKey"/>
                </yfc:makeXMLInput>
                <td>
                    <input type="checkbox" value='<%=getParameter("inboxKey")%>'
name="EntityKey"/>
                </td>
                <td class="tablecolumn">
                    <a <%=getDetailHrefOptions("L01",
getParameter("inboxKey"),"")%> >
                        <yfc:getXMLValue binding="xml:/Inbox/@InboxKey"/>
                    </a>
                </td>
                <td class="tablecolumn"
sortValue="<%=getNumericValue("xml:Inbox:/Inbox/@Priority")%>"><yfc:getXMLValue
```

```
binding="xml:/Inbox/@Priority"/>
</td>
            </tr>
        </yfc:loopXML>
    </tbody>
</table>
```

C.4.20 getParameter

This JSP function obtains the value of the parameter requested from the `pageContext()` function and requests in the following order:

pageContext.getAttribute() -> If not found -> request.getAttribute() -> If not found ->request.getParameter().

This function is typically used to extract the parameters specified while using various Sterling Presentation Framework JSP tags such as `yfc:makeXMLKey` and `yfc:loopXML`.

Syntax
String getParameter(String paramName);

Input Parameters
paramName - Required. Name of the parameter whose value is required.

Example
This example shows how an order list view shows a hyperlinked order number that opens the default detail view. The `yfc:makeXMLInput` JSP tag uses the keys specified to prepare and stores the XML. The XML can be extracted using the `getParameter()` function.

```
<table class="table" editable="false" width="100%" cellspacing="0">
    <thead>
        <tr>
            <td sortable="no" class="checkboxheader">
                <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
            </td>
            <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
        </tr>
    </thead>
```

```
    <tbody>
        <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
            <tr>
                <yfc:makeXMLInput name="orderKey">
                    <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
                </yfc:makeXMLInput>
                <td class="checkboxcolumn">
                   <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
                </td>
                <td class="tablecolumn"><a
href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
                    <yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
                </td>
                <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/></td>
            </tr>
        </yfc:loopXML>
    </tbody>
</table>
```

C.4.21 getRadioOptions

This JSP function XML binds radio buttons when modification rules do not need to be considered.

Syntax

String getRadioOptions(String name)

String getRadioOptions(String name, String checked)

String getRadioOptions(String name, String a_checked, String a_value)

Input Parameters

name - Required. Value of the name attribute for the radio input. Can be a binding or a literal.

checked - Required. Sets the checked attribute for the element with the matching ID. Must be a literal.

value - Required. Value of the value attribute for the radio input. Can be a binding or a literal.

JSP Usage

```
<input type="radio" <%=getRadioOptions("xml:Order:/OrderAddlinfo/@Country"
,"US", "xml:Order:/OrderAddlinfo/@Country" )%>>United States</input>
```

Resultant HTML

```
<input type="radio" name="US" value="US" CHECKED>United States</input>
```

C.4.22 getRequestDOM

This JSP function constructs a YFCElement containing all of the XML elements that could be created out of the request parameters that are available when this function is called. When a particular screen is "posted", the input fields on the screen that are bound to XML values can be accessed via this function. The YFCElement that is constructed contains the following structure:

```
<root>
   <namespace1 ... />
   <namespace2 ... />
   ...
</root>
```

This function is useful when you need to access the posted XML values in the proper XML structure for some JSP processing.

Syntax

String getRequestDOM()

Input Parameters

None.

Output Parameters

A YFCElement containing all of the XMLs that could be created out of the request parameters that were available when this function was called.

JSP Usage

The following example shows the output of getRequestDOM if the following input fields are posted:

```
<input type="hidden" name="xml:/Order/@OrderNo" value="Order0001"/>
<input type="hidden" name="xml:/Order/@OrderType" value="Customer"/>
<input type="hidden" name="xml:/User/@UserName" value="user01"/>
```

Ouput of getRequestDOM is:

```
<root>
    <Order OrderNo="Order0001" OrderType="Customer"/>
    <User UserName="user01"/>
</root>
```

C.4.23 getSearchCriteriaValueWithDefaulting

This JSP function handles the situation where you want to use a default value in the search criteria fields. This special function is required because it is necessary to distinguish between when the user has come to a screen initially versus when a saved search has been loaded and this attribute has specifically been saved as "blank" in the saved search. In that this type of saved search is being loaded into the search view, then the defaulting should not take place.

Syntax

String getSearchCriteriaValueWithDefaulting(String binding, String defaultBinding)

Input Parameters

binding - Required. The target XML binding of the search criteria field that needs to have its value defaulted accordingly.

defaultBinding - Required. The XML binding (or static value) from where the default value should come from when the user is navigating to that search screen for the first time.

Output Parameters

A string containing the value that will be displayed in that search criteria field.

JSP Usage

This example shows how an "enterprise code" combo box on an order search screen can be defaulted to the primary enterprise code of the logged in user's organization.

```
<td class="<%=inputTdClass%>" nowrap="true">
    <select class="combobox" <%=getComboOptions(enterpriseCodeBinding)%>>
<yfc:loopOptions
binding="xml:CommonEnterpriseList:/OrganizationList/@Organization"
name="OrganizationCode"
value="OrganizationCode"
selected='<%=getSearchCriteriaValueWithDefaulting("xml:/Order/@EnterpriseCode",
"xml:CurrentOrganization:/Organization/@PrimaryEnterpriseKey")%>'/>
    </select>
</td>
```

C.4.24 getTextAreaOptions

This JSP function XML binds a text area when modification rules do not need to be considered.

Syntax

getTextAreaOptions(String name)

Input Parameters

name - Required. Value of the `name` attribute for the Text Area Box input. Can be a binding or a literal.

JSP Usage

```
<=%getTexAreaOptions("xml:/Order/Instructions/Instruction/@InstructionText")%>
```

C.4.25 getTextOptions

This JSP function XML binds text input fields when modification rules do not need to be considered.

Syntax

String getTextOptions(String name)

String getTextOptions(String name, String value)

String getTextOptions(String name, String value, String defaultValue)

Inserting this function enables setting the value of a text input when it is displayed and setting the path and attribute in an XML to where the value should go when the form is posted.

If the value and default value are not given, the default value is blanks and value defaults to name.

Input Parameters

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. The target XML is then passed to the appropriate API through the Sterling Service Definition Framework.

value - Required. Specifies what to display as the input text. Can be a binding or a literal. Default is name.

defaultValue - Required. This can be a binding or a literal that is defaulted to in the case that the value binding returns nothing. Default is blanks.

Example

This example results in a input text that displays the value of a bound attribute and sets the value of a bound attribute when the form is posted. This is the XML referenced by the bindings:

```
<Order>
    <addlInfo Country="US"></addlInfo>
</Order>
```

JSP Usage

```
<input type="text" <%=getTextOptions("xml:Order:/OrderAddlinfo/@Country"
,"xml:Order:/OrderAddlinfo/@Country", "USA")%> />
```

HTML Results

When the value binding is found:

```
<input type="text" name=" xml:Order:/OrderAddlinfo/@Country " value="US"
Datatype='' size="10" Decimal='' OldValue="United States"/>
```

When the name binding is not found:

```
<input type="text" name="US" value="USA" Datatype='' size="10" Decimal=''
OldValue="United States"/>
```

Using getTextOptions within an editable list:

- An underscore ("_") and a counter must be appended to the first parameter of getTextOptions.

- The name of the counter is the value of the ID attribute specified in the loopXML tag. Set the ID attribute to be the same as the name of the child node on which you are looping.

Example

Inserting a Text Box.

```
<input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/Order/OrderLines/OrderLine_" + OrderLineCounter +
"/@ShipNode", "xml:/OrderLine/@ShipNode")%>/>
```

C.4.26 getUITableSize

This JSP function returns the UI width of the attribute passed as input. This can be used to set the column width of tables within your screens to achieve consistent sizing of the columns throughout the application. The width that is used comes from the data type definition of the attribute. See the extending datatypes.

Syntax

getUITableSize(String binding)

Parameters

Path to the current attribute in XML.

Note: this should be used in the style attribute of all <td> tags within all <thead> tags of list tables.

JSP Usage

```
style="width:<%=getUITableSize("xml:/Order/@OrderDate")%>"
```

C.4.27 getValue

This JSP function gets the value of the binding string provided from the XML namespace provided.

Syntax
String getValue(String xmlName,String binding)

Input Parameters
xmlName - Required. Namespace of the XML. Even if the binding string contains the namespace, this must be provided.

binding - Required. Binding string.

Example
This example shows how the supply type is extracted from the output of the API in the Inventory Adjustment screen.

```
<%
    String
supplyType=getValue("Item","xml:/Item/Supplies/InventorySupply/@SupplyType");
%>
```

C.4.28 getValue - Overloaded

Deprecated in Sterling Supply Chain Applications release 4.5. Replaced by "getValue" on page 622. This JSP function gets the value of the binding string provided from the YFCElement.

Syntax
String getValue(YFCElement element, String binding)

Input Parameters
element - Required. YFCElement from which the attribute must be extracted.

binding - Required. Binding string.

Example
This example shows how the supply type could be extracted from the output of the API in the Inventory Adjustment screen.

```
<%
    YFCElement elem=getParameter("Item");
    String
supplyType=getValue(elem,"xml:/Item/Supplies/InventorySupply/@SupplyType");
%>
```

C.4.29 goToDetailView

This JSP function can be used to conditionally display different detail views based on the logic specified within a JSP page. This function can only be used in conjunction with detail views that have been defined as "redirector views". This function is useful when you need to conditionally navigation to a different detail view based on some logic (possibly determined by the output of an API call). In the JSP anchor page of a redirector view, use this function to ultimately navigate to the detail view that will be shown to the user.

Syntax

void goToDetailView(HttpServletResponseresponse, String viewGroupId)

Input Parameters

response - Required. The response object. Pass the "response" object as is from your redirector JSP.

viewGroupId - Required. The view group ID that will be shown to the end user.

Output Parameters

None.

JSP Usage

This example shows the complete JSP that is the anchor page of the shipment detail redirector view. If the shipment that will be displayed is a shipment for a provided service, then a different detail view is displayed. Note that the output of the getShipmentDetails() API is used to determine which view should be displayed.

```
<%@include file="/yfsjspcommon/yfsutil.jspf"%>
<%
    String sViewGrp = "YOMD710";
    if (isTrue("xml:/Shipment/@IsProvidedService")) {
        sViewGrp = "YOMD333";
    }
    goToDetailView(response, sViewGrp);
%>
```

C.4.30 isModificationAllowed

This JSP function is used to determine if modification is permitted for a certain attribute for the current entity.

Syntax

boolean isModificationAllowed(String name, String allowModBinding)

Input Parameters

name - Required. Path in the target XML attribute. If this attribute is modifiable for the current entity's status, the function returns *true*. If it is not modifiable, the function returns *false*.

allowModBinding - Required. Binding string that points to a set of elements containing modification types that are permitted for the current status.

JSP Usage

This example shows how the table footer containing the dynamic add rows feature can be included in a page based on whether or not add rows is permitted for the current order.

```
    <%if
(isModificationAllowed("xml:/@AddInstruction","xml:/Order/AllowedModifications")
) {%>
    <tr>
    <td nowrap="true" colspan="3">
    <jsp:include page="/common/editabletbl.jsp" >
    </jsp:include>
    </td>
    </tr>
    <%}%>
```

C.4.31 isPopupWindow

This JSP function determines whether or not the current window is displayed in a popup window. Use this function when the logic in your screen must differ when it appears in a popup window.

Syntax

isPopupWindow()

Input Parameters
None.

Output Parameters
A boolean indicating whether or not the current window is displayed in a popup window.

JSP Usage
In this example, the selected value that appears in the combobox is different depending on whether this screen is being shown within a popup window.

```
<select name="xml:/Shipment/@EnterpriseCode" class="combobox">
   <% if (isPopupWindow()) { %>
     <yfc:loopOptions
     binding="xml:EnterpriseList:/OrganizationList/@Organization"
     name="OrganizationCode"
     value="OrganizationCode" selected="xml:/Shipment/@EnterpriseCode" />
   <% } else { %>
     <yfc:loopOptions
     binding="xml:EnterpriseList:/OrganizationList/@Organization"
     name="OrganizationCode"
     value="OrganizationCode"
     selected='<%=getSelectedValue("xml:/Shipment/@EnterpriseCode")%>' />
   <% } %>
</select>
```

C.4.32 isTrue

This JSP function returns *true* if the attribute specified in the input parameter has a value of Y, or *true*. Otherwise, it returns *false*. It is not case sensitive.

Syntax
boolean isTrue(String bindingStr);

Input Parameters
bindingStr - Required. Binding string that specifies which attribute to evaluate.

Output Parameters

A boolean indicating if the attribute being evaluated has a value of Y or *true*.

Example

This example uses the `isTrue()` function to find out the value of the attribute referred to in the `xml:/OrderRelease/@HasDerivedParent` binding.

```
boolean isAgainstOrder=isTrue("xml:/OrderRelease/@HasDerivedParent");
```

C.4.33 isVoid

This JSP function determines whether the object passed is null or contains only white spaces.

Syntax

boolean isVoid(Object obj)

Input Parameters

obj - Required. Object that must be checked for null or white spaces.

Example

This example shows how this function is used to check if a specific attribute is void.

```
<% if (!isVoid(getParameter("ShowShipNode"))) {%>
<tr>
    <td class="detaillabel" ><yfc:i18n>Ship_Node</yfc:i18n></td>
    <td class="protectedtext"><yfc:getXMLValue
binding="xml:/InventoryInformation/Item/@ShipNode"
name="InventoryInformation"></yfc:getXMLValue></td>
</tr>
<%}%>
```

C.4.34 resolveValue

This JSP function gets the value of the binding string provided from the `YFCElement` provided.

Syntax

String resolveValue(String binding)

Input Parameters

binding - Required. Binding string. Binding string can contain the namespace.

Example

This example shows how this function is used to resolve the value pointed to by a binding string.

```
<%
String reqshipdate=resolveValue("xml:OrderEntry:/Order/@ReqShipDate");
%>
```

C.4.35 showEncryptedCreditCardNo

This JSP function returns a value to the display that represents an encrypted credit card number.

Syntax

showEncryptedCreditCardNo(String CreditCardNo)

Input Parameters

CreditCardNo - Required. String containing the last four digits of a credit card number.

Example

```
<%=showEncryptedCreditCardNo(resolveValue("xml:/PaymentMethod/@Disp
layCreditCardNo"))%>
```

C.4.36 userHasOverridePermissions

This JSP function determines whether or not the current login user has permission to override the modifications rules configuration.

Syntax

boolean userHasOverridePermissions()

C.4.37 yfsGetCheckBoxOptions

This JSP function XML binds checkboxes when modification rules need to be considered.

Syntax

String yfsGetCheckBoxOptions(String name,String a_checked, String a_ value, String allowModBinding)

Input Parameters

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. Through the Sterling Service Definition Framework, the target XML is then passed to the appropriate API.

checked - Required. When the value of the value attribute is equal to this value, the `checked` attribute is set to true.

value - Required. Value of the `value` attribute for the checkbox input. Can be a binding or a literal.

allowModBinding - Required. Binding string that points to a set of elements containing modification types that are permitted for the current status.

JSP Usage

```
<input class="checkbox" type="checkbox"
<%=yfsGetCheckBoxOptions("xml:/Order/@ChargeActualFreightFlag","xml:/Order/@Char
geActualFreightFlag","Y","xml:/ Order/AllowedModifications")%>/>
```

C.4.38 yfsGetComboOptions

This JSP function XML binds combo boxes when modification rules need to be considered.

Syntax

String yfsGetComboOptions(String name, String allowModBinding)

String yfsGetComboOptions(String name, String value, String allowModBinding)

Input Parameters

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. Through the Sterling Service Definition Framework, the target XML is then passed to the appropriate API.

value - Required. Specifies what to display as the input text. Can be a binding or a literal.

allowModBinding - Required. Binding string that points to a set of elements containing modification types that are permitted for the current status.

JSP Usage

```
<select <% if (isVoid(modifyView)) {%> <%=getProtectedComboOptions()%> <%}%>
<%=yfsGetComboOptions("xml:/Order/@ScacAndServiceKey",
"xml:/Order/AllowedModifications")%>>
    <yfc:loopOptions binding="xml:/ScacAndServiceList/@ScacAndService"
name="ScacAndServiceDesc"
    value="ScacAndServiceKey" selected="xml:/Order/@ScacAndServiceKey"/>
</select>
```

C.4.39 yfsGetImageOptions

This JSP function builds an image tag in HTML. The image may be *hidden*, based on whether the modification of the XML attribute passed as a parameter is permitted or not, unlike the getImageOptions() function. See "getImageOptions" on page 610.

A Java constants file keeps the image path and icon centralized. If the path starts with /yantra/console/icons, the image file is first searched for inside /webpages/extn/icons/yantraiconsbe.jar (or the localized icons JAR file) and then inside the /webpages/yfscommon/yantraiconsbe.jar (or the localized icons JAR file). The path to be specified is the path of the image file inside the JAR file.

If the path does not start with /yantra/console/icons, it picks up the file from the location in the EAR file. It is strongly advised that you place your images under /console/icons in the custom icons JAR file (yantraiconsbe.jar).

The path to be specified is the path of the image file inside the JAR file.

Syntax

String yfsGetImageOptions(String src, String alt, String name, String allowModBinding)

Parameters

src - Required. Image file name, including the path, within the icons JAR file.

alt - Required. Tooltip to use for the image.

name - Required. Path in the target XML attribute. This function shows the image only when modification of this attribute is permitted based on the status of the current entity.

allowModBinding - Required. Binding string that points to a set of elements containing modification types that are permitted for the current order status.

JSP Usage

```
<img class="lookupicon" name="search" onclick="invokeCalendar(this);return
false"  <%=yfsGetImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar",
"xml:/Order/@ReqShipDate",  "xml:/Order/AllowedModifications")%>/>
```

C.4.40 yfsGetTemplateRowOptions

This JSP function XML binds input fields when the field appears within an editable table's template row. The template row appears when the plus icon ("+") is selected in an editable table.

Syntax

String yfsGetTemplateRowOptions(String name, String allowModBinding, String modType, String controlType)

Input Parameters

name - Required. Value of the name attribute for the input. Can be a binding or a literal.

allowModBinding - Required. Binding string that resolves to a list of elements containing all modification types permitted for the current status of the entity.

modType - Required. Modification type associated with the current control.

controlType - Required. Type of control. Can be a textbox, checkbox or textarea.

JSP Usage

```
<input type="text" <%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_
/Item/@ItemID",  "xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
```

Example

This example shows how this function is used to store a template row in the list of order lines for an order in the Order detail view.

```
    <tfoot>
        <tr style='display:none' TemplateRow="true">
            <td class="checkboxcolumn">
                <input type="hidden"
<%=getTextOptions("xml:/Order/OrderLines/OrderLine_/@Action", "",  "CREATE")%>
/>
            </td>
            <td class="tablecolumn"> </td>
            <td class="tablecolumn"> </td>
            <td class="tablecolumn" nowrap="true">
                <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/Item/@ItemID",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
                <img class="lookupicon"
onclick="templateRowCallItemLookup(this,'ItemID','ProductClass','UnitOfMeasure',
'item')"  <%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON, "Search_for_
Item")%>/>
            </td>
            <td class="tablecolumn">
                <select
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_
/Item/@ProductClass",  "xml:/Order/AllowedModifications", "ADD_LINE",
"combo")%>>
                    <yfc:loopOptions
binding="xml:ProductClassList:/CommonCodeList/@CommonCode"  name="CodeValue"
                    value="CodeValue"
selected="xml:/Order/OrderLine/Item/@ProductClass"/>
                </select>
            </td>
            <td class="tablecolumn">
```

```
                    <select
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_
/Item/@UnitOfMeasure",  "xml:/Order/AllowedModifications", "ADD_LINE",
"combo")%>>
                    <yfc:loopOptions
binding="xml:UnitOfMeasureList:/CommonCodeList/@CommonCode"  name="CodeValue"
                    value="CodeValue"
selected="xml:/Order/OrderLine/Item/@UnitOfMeasure"/>
                    </select>
            </td>
            <td class="tablecolumn"> </td>
            <td class="tablecolumn" nowrap="true">
                    <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@ReceivingNode",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
                    <img class="lookupicon" onclick="callLookup(this,'shipnode')"
<%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON, "Search_for_Recieving_
Node")%>/>
            </td>
            <td class="tablecolumn" nowrap="true">
                    <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@ShipNode",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
                    <img class="lookupicon" onclick="callLookup(this,'shipnode')"
<%=getImageOptions(YFSUIBackendConsts.LOOKUP_ICON, "Search_for_Ship_Node")%>/>
            </td>
            <td class="tablecolumn" nowrap="true">
                    <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@ReqShipDate",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>/>
                    <img class="lookupicon" onclick="invokeCalendar(this)"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar")%>/>
            </td>
            <td class="numerictablecolumn">
                    <input type="text"
<%=yfsGetTemplateRowOptions("xml:/Order/OrderLines/OrderLine_/@OrderedQty",
"xml:/Order/AllowedModifications", "ADD_LINE", "text")%>>
            </td>
            <td class="tablecolumn"> </td>
            <td class="tablecolumn"> </td>
        </tr>
<%if (isModificationAllowed("xml:/@AddLine","xml:/Order/AllowedModifications"))
{ %>
        <tr>
        <td nowrap="true" colspan="13">
```

```
            <jsp:include page="/common/editabletbl.jsp" >
            </jsp:include>
            </td>
            </tr>
            <%}%>
        </tfoot>
```

C.4.41 yfsGetTextAreaOptions

This JSP function XML binds text areas when modification rules need to be considered.

Syntax

String yfsGetTextAreaOptions(String name, String a_value, String allowModBinding)

String yfsGetTextAreaOptions(String name, String allowModBinding)

Parameters

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. Through the Sterling Service Definition Framework, the target XML is then passed to the appropriate API.

value - Required. Specifies what to display as the input text. Can be a binding or a literal.

allowModBinding - Required. Binding string that points to a set of elements containing modification types that are permitted for the current status.

JSP Usage

```
<textarea class="unprotectedtextareainput" rows="3" cols="100"
<%=yfsGetTextAreaOptions("xml:/Order/Instructions/Instruction_" +
InstructionCounter +  "/@InstructionText","xml:/Instruction/@InstructionText",
"xml:/Order/AllowedModifications")%>><yfc:getXMLValue
binding="xml:/Instruction/@InstructionText"/></textarea>
```

C.4.42 yfsGetTextOptions

This JSP function XML binds text input fields when modification rules need to be considered.

Syntax

String yfsGetTextOptions(String name, String allowModBinding)

String yfsGetTextOptions(String name, String value, String allowModBinding)

String yfsGetTextOptions(String name, String value, String defaultValue, String allowModBinding)

Input Parameters

name - Required. Path in the target XML to which the value in the input text is sent when the form is posted. Through the Sterling Service Definition Framework, the target XML is then passed to the appropriate API.

value - Required. Specifies what to display as the input text. Can be a binding or a literal.

defaultValue - Required. This can be a binding or a literal that is defaulted to in the case that the value binding returns nothing.

allowModBinding - Required. This is a binding string that points to a set of elements containing modification types that are permitted for the current status.

JSP Usage

```
<input type="text" <%=yfsGetTextOptions("xml:/Order/@ReqShipDate",
"xml:/Order/AllowedModifications")%>/>
```

C.5 JSP Tag Library

Use these tags that are supplied by Sterling Supply Chain Applications to serve your dynamic web pages.

C.5.1 callApi

The callApi JSP tag calls an API from within the JSP file. In most cases, it is not necessary to make an API call from inside a JSP file. However, occasionally there is no other option. For example, when an API must be called multiple times within a loop, use the callApi JSP tag.

When you use this JSP tag on a view, you may enable the Skip Automatic Execution checkbox on the Resource configuration screen for the API Resource that you intend to call. This prevents the API from being called when the view is initially opened. This option is *not* available for API resources that are created directly under an entity resource.

Attributes

apiID - Required. Postfix of the resource ID of the API to be called. When an API resource is configured through the Resource Configurator, a postfix must be supplied for the resource ID. This is the postfix value that must be used.

Body

None.

Example

In this example, the callAPI is used to retrieve additional attributes about an item using the `getItemDetails()` API defined in the API resource containing the API in the ID. Note that the API input or template is not specified anywhere in the JSP. This is configured in the API resource definition just like every other API.

```
<yfc:loopXML binding="xml:/OrderLineStatusList/@OrderStatus" id="OrderStatus">
    <tr>
        <yfc:makeXMLInput name="orderLineKey">
            <yfc:makeXMLKey binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderStatus/OrderLine/@OrderLineKey"/>
  <yfc:callAPI apiID='AP1'/>
  < ... >
        </tr>
```

After the `callApi` JSP tag is used in the JSP, the output is available in the corresponding output namespace.

C.5.2 callAPI (Alternative Method)

The callAPI JSP tag also supports a way to call APIs within JSPs without defining the API in the Resource Configurator. If called in this way, different attributes needs to be passed as input to the tag. This alternative method should be used when the input or template of an API call needs to be dynamic based on some conditions within the JSP. Additionally, this alternative method may be used if the input to the API is complicated and cannot be formed using the traditional techniques.

Attributes

apiName - Optional. The name of the API that will be called. When using this alternative method of callAPI either apiName **or** serviceName is required.

serviceName - Optional. The name of the service (from Sterling Service Definition Framework) that will be called. Calling a service does not support passing templates. When using this alternative method of callAPI, either serviceName **or** apiName is required.

inputElement - Required when using this alternative method. A YFCElement representing the input element that is to be passed to the API.

templateElement - Conditionally required. A YFCElement representing the output template expected for the API. When using this alternative method, if the apiName attribute is used, then templateElement is required. If serviceName is used, then templateElement is ignored.

outputNamespace - Optional. The namespace under which the output of the API will be placed.

The output of the API is saved in this namespace. Namespace is optional, but if it is not specified, it is defaulted to the root node name of the XML under consideration. Therefore, while referring to the output of the API, even if namespace is not specified here, it can be assumed to be the same as the root node name of the output.

A namespace is a tag that can be used to identify a specific XML. The Sterling Presentation Framework enables you to call multiple APIs and

store the outputs in different namespaces. In your JSP or in the input to an API, you can refer to values from any namespace that is available at that point.

inputNamespace - Optional. The input namespace is used to dynamically resolve additional input to the API. For more information about input namespace see Section 3.25, "API Input" on page 96.

Body

None.

Example

The following example shows how the getOrderDetails() API can be called from within a JSP without defining an API resource in the Resource Configurator. Note how the input and template elements are formed in the JSP before the callAPI tag is used. After the callAPI tag call, the output of the getOrderDetails() API is available in the RelatedFromOrderDetails namespace that can be used later on within the JSP.

```
<%
   YFCDocument inputDoc = YFCDocument.parse("<Order
OrderHeaderKey=\"xml:/Document/@RelatedFromOrderHeaderKey\"/>");
YFCDocument templateDoc = YFCDocument.parse("<Order EnterpriseCode=\"\"
OrderHeaderKey=\"\" OrderNo=\"\"
   Status=\"\" BuyerOrganizationCode=\"\" SellerOrganizationCode=\"\"
OrderDate=\"\" RulesetKey=\"\" HoldFlag=\"\" DocumentType=\"\"
isHistory=\"\"/>");
%>
<yfc:callAPI apiName='getOrderDetails'
inputElement='<%=inputDoc.getDocumentElement()%>'
templateElement='<%=templateDoc.getDocumentElement()%>'
outputNamespace='RelatedFromOrderDetails'/>
```

C.5.3 getXMLValue

The getXMLValue JSP tag returns the value of an XML attribute specific to an XML binding.

Attributes

name - Optional. String containing the namespace of the XML from which a value must be obtained. If this parameter is not used, the value

is picked up from the binding. For example, if you specify `xml:/Menu/@MenuDescription` as the binding, the value for name defaults to *Menu*. Or in another example, if you specify `xml:/mymenu:/Menu/@MenuDescription` as the binding, the name defaults to *mymenu*.

binding - Required. String containing the XML path that points to the attribute of the requested value.

Body
None.

Example
```
<td class="protectedtext"><yfc:getXMLValue binding="xml:/Category/@CategoryID"
name="Category" /></td>
```

C.5.4 getXMLValueI18NDB

The getXMLValueI18NDB JSP tag returns the localized value of an XML attribute specific to an XML binding based on the user's locale.

Attributes
name - Optional. String containing the namespace of the XML from which a value must be obtained. If this parameter is not used, the value is picked up from the binding. For example, if you specify `xml:/Menu/@MenuDescription` as the binding, the value for name defaults to *Menu*. Or in another example, if you specify `xml:/mymenu:/Menu/@MenuDescription` as the binding, the name defaults to *mymenu*.

binding - Required. String containing the XML path that points to the attribute of the requested value.

Body
None.

Example
```
<td class="protectedtext"><yfc:getXMLValueI18NDB
binding="xml:/Category/@Description" name="Category" /></td>
```

C.5.5 hasXMLNode

The hasXMLNode JSP tag is used to determine if a specific XML element or attribute is returned by the API.

Attributes
binding - Required. String containing the XML path of the element or attribute to seek. If the binding string contains an attribute, this tag does not permit its body to be processed if the attribute is void, even if the element exists.

Body
Can contain HTML that is written only if the hasXMLNode evaluates to true.

Example
This example shows how a kit icon is shown for those lines belonging to an order release that contain kits.

```
<td class="tablecolumn" nowrap="true">
            <yfc:hasXMLNode binding="xml:/OrderLine/KitLines/KitLine">
                <a <%=getDetailHrefOptions("L03",
getParameter("orderLineKey"), "")%>>
                       <img class="columnicon"
<%=getImageOptions(YFSUIBackendConsts.KIT_COMPONENTS_COLUMN,  "Kit_
Components")%>>
                </a>
            </yfc:hasXMLNode>
        </td>
```

This example shows how a parent kit line icon is shown for those lines that have a parent kit line.

```
<yfc:hasXMLNode binding="xml:/OrderLine/@OrigOrderLineKey">
<a <%=getDetailHrefOptions("L05", getParameter("origOrderLineKey"), "")%>>
<img class="columnicon" <%=getImageOptions(YFSUIBackendConsts.DERIVED_
ORDERLINES_COLUMN, "Kit_Parent_Line")%>>
</a>
</yfc:hasXMLNode>
```

C.5.6 i18n

The i18n JSP tag retrieves the localized description of the key from the resource bundles. Use this for all literals in the HTML.

Attributes
None.

Body
Key that must be resolved into the localized string. For more information on how the system uses locale-specific resource bundles, see the *Sterling Supply Chain Applications Localization Guide*.

Example
This example shows how query types (is, starts with, contains) are shown in a select tag from the output of an API.

```
<tr>
    <td class="searchlabel" ><yfc:i18n>Product_Class</yfc:i18n></td>
</tr>
```

C.5.7 i18ndb

The i18ndb JSP tag retrieves the localized description of the value from the YFS_LOCALIZED_STRING table based on the user's locale. Use this to get the localized database descriptions in the HTML.

Attributes
None.

Body
Key that must be resolved into the localized string. For more information on how the system uses locale-specific resource bundles, see the *Sterling Supply Chain Applications Localization Guide*.

Example
This example shows how query types (is, starts with, contains) are shown in a select tag from the output of an API.

```
<tr>
    <td>
```

```
            <yfc:i18ndb><%=resolveValue("xml:/Shipment/Status/@StatusName")%>
            </yfc:i18ndb>
        </td>
    </tr>
```

C.5.8 loopOptions

The loopOptions JSP tag builds the options belonging to the HTML select tag.

Attributes

binding - Required. Binding string that points to the repeating element in the API output. The repeating element must be one fixed with the at character ("@").

name - Optional. Attribute name within the binding element to be used for the description visible to the user in the option tag. If not passed, defaults to *name*, which means that Sterling Supply Chain Applications search for an attribute called *name*.

value - Optional. Attribute name within the binding element to be used for the value attribute of the option tag. If not passed, it defaults to *value*, which means that Sterling Supply Chain Applications search for an attribute called *value*.

selected - Optional. Binding string that must be evaluated and set as the default selected value. This is matched with the value attribute, not the description attribute. Defaults to blanks, for example, space (" ").

isLocalized - Optional. If passed as "Y" it obtains the localized description to be displayed based on the user's locale from the YFS_LOCALIZED_STRINGS table.

targetBinding - Optional. If the target binding of the select is different than the source binding, you must specify the target binding as input when using loopOptions. This will ensure that data entered by the end user will not be lost even when an exception is generated in the API.

Body
None.

Examples

This example shows how query types (is, starts with, contains) are shown in a select tag from the output of an API.

```
    <td nowrap="true" class="searchcriteriacell" >
        <select name="xml:/Item/@ItemIDQryType" class="combobox" >
            <yfc:loopOptions
binding="xml:/QueryTypeList/StringQueryTypes/@QueryType"
                name="QueryTypeDesc" value="QueryType"
selected="xml:/Item/@ItemIDQryType"/>
        </select>
        <input type="text" class="unprotectedinput"
<%=getTextOptions("xml:/Item/@ItemID") %> />
    </td>
```

This example uses combo boxes within an editable list:

- An underscore character ("_") and a counter must be appended to the name attribute of the select element.

- The name of the counter is the value of the ID attribute specified in the loopXML tag. The ID attribute should always be set to the same as the child node name on which you are looping.

```
<select name="xml:/Order/Instructions/Instruction_
<%=InstructionCounter%>/@InstructionType" class="combobox">
<yfc:loopOptions binding="xml:InstructionTypeList:/CommonCodeList/@CommonCode"
name="CodeShortDescription"
           value="CodeValue" selected="xml:/Instruction/@InstructionType"/>
</select>
```

C.5.9 loopXML

The loopXML JSP tag loops through a specific repeating element in a source XML.

> **Note:** If your application server only supports up to JSP specification version 1.1, it does not support using `jsp:include` within a custom JSP tag that contains a body tag. Using the `loopXML` tag results in a run-time JSP error that indicates "Illegal to flush within a custom tag".
>
> To avoid this run-time error, use the `getLoopingElementList()` function instead of the `loopXML` tag. See "getLoopingElementList" on page 613.

Attributes

binding - Required. Path to the element through which you want to loop within the source XML. The repeating element must be one fixed with the at character ("@").

name - Optional. Name of the source XML. If this parameter is not used, the value is picked up from the binding. For example, if you specify `xml:/Menu/@MenuDescription` as the binding, the value for name defaults to *Menu*. Or in another example, if you specify `xml:/mymenu:/Menu/@MenuDescription` as the binding, the `name` defaults to *mymenu*.

id - Optional. Name of the created YFCElement that holds the element resolved from the binding. If not specified, the Element NodeName pointed to by the binding parameter is used. For example, if the binding is `xml:/ItemList/@Item` and is not passed, the value for `id` defaults to *Item*.

Body
Can contain HTML that is written for each iteration in the loop.

Example
This example shows how the `loopXML` JSP tag is used to display the list of items in item lookup.

```
<tbody>
    <yfc:loopXML name="ItemList" binding="xml:/ItemList/@Item" id="item">
    <tr>
        <td class="tablecolumn">
            <img class="icon"
onclick="setItemLookupValue('<%=resolveValue("xml:item:/Item/@ItemID")%>','<%=re
```

```
solveValue("xml:item:/Item/Prim
aryInformation/@DefaultProductClass")%>','<%=resolveValue("xml:item:/Item/@UnitO
fMeasure")%>')"    value="<%=resolveValue("xml:item:/Item/@ItemID")%>"
<%=getImageOptions(YFSUIBackendConsts.GO_ICON,"Click_to_select")%> />
        </td>
        <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/@ItemID"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/PrimaryInformation/@DefaultProductClass"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/@UnitOfMeasure"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/PrimaryInformation/@ShortDescription"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/PrimaryInformation/@MasterCatalogID"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="item"
binding="xml:/Item/@OrganizationCode"/></td>
    </tr>
    </yfc:loopXML>
</tbody>
```

C.5.10 makeXMLInput

The makeXMLInput JSP tag is used in conjunction with makeXMLKey to form a hidden key that is used to pass data from list to detail screens.

Attributes
name - Required. The name of the hidden input HTML tag that gets formed as a result of this JSP tag.

Body
Can contain multiple makeXMLKey JSP tags. The output of the makeXMLKey JSP tags are concatenated into a single hidden input.

Example
This example shows how this JSP tag is used in conjunction with the makeXMLKey JSP tag to form a hidden input to pass inventory key data from the Inventory list view to the Inventory detail view.

```
<tbody>
    <yfc:loopXML name="InventoryList"
binding="xml:/InventoryList/@InventoryItem" id="InventoryItem"
```

```
keyName="InventoryItemKey" >
    <tr>
        <yfc:makeXMLInput name="inventoryItemKey">
            <yfc:makeXMLKey binding="xml:/InventoryItem/@ItemID"
value="xml:/InventoryItem/@ItemID" />
            <yfc:makeXMLKey binding="xml:/InventoryItem/@UnitOfMeasure"
value="xml:/InventoryItem/@UnitOfMeasure" />
            <yfc:makeXMLKey binding="xml:/InventoryItem/@ProductClass"
value="xml:/InventoryItem/@ProductClass"  />
            <yfc:makeXMLKey binding="xml:/InventoryItem/@OrganizationCode"
value="xml:InventoryList:/InventoryList/@OrganizationCode" />
        </yfc:makeXMLInput>
        <td class="checkboxcolumn">
            <input type="checkbox" value='<%=getParameter("inventoryItemKey")%>'
name="EntityKey"/>
        </td>
        <td class="tablecolumn">
            <a
href="javascript:showDetailFor('<%=getParameter("inventoryItemKey")%>');"><yfc:g
etXMLValue  name="InventoryItem" binding="xml:/InventoryItem/@ItemID"/></a>
        </td>
        <td class="tablecolumn"><yfc:getXMLValue name="InventoryItem"
binding="xml:/InventoryItem/@ProductClass"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="InventoryItem"
binding="xml:/InventoryItem/@UnitOfMeasure"/></td>
        <td class="tablecolumn"><yfc:getXMLValue name="InventoryItem"
binding="xml:/InventoryItem/Item/PrimaryInformation/@Description"/></td>
    </tr>
    </yfc:loopXML>
</tbody>
```

C.5.11 makeXMLKey

The makeXMLKey JSP tag is used in conjunction with makeXMLInput to form a hidden key that is used to pass data from list to detail screens.

Attributes
binding - Required. The binding string that must be resolved and stored in the hidden input to be passed on to the detail screen.

Body
None.

Example

Refer to the example in "makeXMLInput" on page 645.

C.6 JavaScript Functions

The Sterling Supply Chain Applications UI uses JavaScript functions to perform client-side operations such as opening popup windows, switching views, validating user input, and so forth. Most of the JavaScripts used in the UI are provided by the UI infrastructure layer. You can use the same functions while performing your UI extensions. This section describes JavaScript functions supplied by Sterling Supply Chain Applications's UI layer.

Note that Sterling Supply Chain Applications also use JavaScript functions that are not supplied by the UI infrastructure. These functions usually perform some specific action for some specific screen and are not required to be used during your UI extensions.

In addition, if you require additional logic for your screen for which Sterling Supply Chain Applications UI infrastructure does not provide JavaScript functions, you can write and use your own as needed.

Lookup

callLookup - see page 648. Uses [GET]
invokeCalendar - see page 660. Uses [GET]
yfcShowSearchPopup - see page 694. Uses [GET]

Control Name

ignoreChangeNames - see page 659
yfcDoNotPromptForChanges - see page 674
yfcDoNotPromptForChangesForActions - see page 675
yfcHasControlChanged - see page 678
yfcSetControlAsUnchanged - see page 683
yfcSpecialChangeNames - see page 695

Event Handler

validateControlValues - see page 667
yfcBodyOnLoad - see page 669
yfcGetSaveSearchHandle - see page 676

Show Detail

Show List Popup

Other

C.6.1 callLookup

This JavaScript function displays a lookup screen that enables the user to search for and select a record to use in the current screen. For example, an organization lookup on the order entry screen enables the user to select a buyer organization. Typically, you should attach this function to the onclick event of an image within your JSP page.

Syntax

callLookup(obj,entityname,extraParams)

Input Parameters

entityname - Optional. Entity to search for in the lookup screen. If not passed, defaults to the name of the current entity.

obj - Required. Handle to the image being selected.

extraParams - Optional. Passes extra parameters to the lookup screen. The format of the parameter is name/value pairs in URL format. If passed, the parameters are passed to the lookup screen.

Output Parameters

None.

Example

This example shows how to show an organization lookup that defaults the display of the seller role in the buyer Role field on the lookup:

```
<img class="lookupicon"
onclick="callLookup(this,'organization','xml:/Organization/OrgRoleList/OrgRole/@
RoleKey=BUYER')" name="search" <%=getImageOptions(YFSUIBackendConsts.LOOKUP_
ICON, "Search_for_Buyer") %> />
```

C.6.2 doCheckAll

This JavaScript function toggles the state of all the checkboxes in a table, using the following assumptions:

- The table must have separate head and body sections.

- Checkboxes within the body section must have the same column index as the specified checkbox object. Cells containing multiple checkboxes are all toggled.

Syntax

doCheckAll(obj)

Input Parameters

obj - Optional. Handle to the checkbox object (in HTML object hierarchy) on a table header. If the object is not passed, the function just returns.

Return Values

None.

Example

This example shows how an order list view showing order number and enterprise would handle the check all and uncheck all option in the table header row.

```
<table class="table" editable="false" width="100%" cellspacing="0">
<thead>
<tr>
<td sortable="no" class="checkboxheader">
<input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
</td>
<td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
<td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
</tr>
</thead>
<tbody>
<yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
<tr>
<yfc:makeXMLInput name="orderKey">
<yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
</yfc:makeXMLInput>
                <td class="checkboxcolumn">
<input type="checkbox" value='<%=getParameter("orderKey")%>' name="EntityKey"/>
</td>
<td class="tablecolumn"><a
href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
<yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
</td>
<td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/></td>
</tr>
</yfc:loopXML>
</tbody>
</table>
```

C.6.3 doCheckFirstLevel

This JavaScript function is used on the onclick event of a checkbox in the table column header. The function checks or unchecks all checkboxes in the *first* level of checkboxes in the table. This function is very similar to the doCheckAll JavaScript function, except that doCheckAll checks or unchecks *all* checkboxes within the specified HTML table.

Use this function for HTML tables the require this "(un)check all" functionality and also have one or more unrelated checkboxes inside the table that should not be affected by the selection checkboxes.

Syntax

doCheckFirstLevel(obj)

Input Parameters

obj - Optional. Handle to the checkbox object (in HTML object hierarchy) on a table header. If the object is not passed, the function does nothing.

Output Parameters

None.

Example

This example shows the table header definition for a list of items containing a checkbox where the user can select one or more items in the table. The header row of the table contains a checkbox. When this checkbox is selected, all of the first level check boxes within the HTML table are checked or unchecked.

```
<table class="table" cellspacing="0" width="100%">
    <thead>
        <tr>
            <td class="checkboxheader" sortable="no" style="width:10px">
                <input type="checkbox" value="checkbox" name="checkbox"
onclick="doCheckFirstLevel(this);"/>
            </td>
            <td class="tablecolumnheader"
style="width:30px"><yfc:i18n>Options</yfc:i18n></td>
            <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/AdditionalS
erviceItems/Item/@ItemID")%>"><yfc:i18n>Item_ID</yfc:i18n></td>
            <td class="tablecolumnheader"
```

```
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/AdditionalS
erviceItems/Item/@UnitOfMeasure")%>"><yfc:i18n>UOM</yfc:i18n></td>
            <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/AdditionalS
erviceItems/Item/PrimaryInformation/@Description")%>"><yfc:i18n>Item_
Description</yfc:i18n></td>
            <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/AdditionalS
erviceItems/Item/@Price")%>"><yfc:i18n>Price</yfc:i18n></td>
        </tr>
    </thead>
```

C.6.4 expandCollapseDetails

This JavaScript function toggles the display state of the specified tags that have expanded and collapsed views.

Syntax

expandCollapseDetails(div_id, expandAlt, collapseAlt, expandgif, collapsegif)

Input Parameters

div_id - Required. Identifier of the object to expand or collapse.

expandAlt - Required. Tooltip to show for expanding a selection. This tooltip shows when the object is in a collapsed state.

collapseAlt - Required. Tooltip to show for collapsing a selection. Available when the object is an expanded state.

expandgif - Required. Image to show when the selection is in a collapsed state.

collapsegif - Required. Image to show when the selection is in an expanded state.

Return Value

None.

Example

This example shows how the expandCollapseDetails() function can be used in a table to hide some advanced information that the user can

retrieve by selecting a special icon at line level. The example shows how payment collection details, such as credit card number, can be viewed by selecting the plus (+) icon. The example also shows `div`, which enables you to specify whether to hide or show information. By default, the `div` is hidden (`display:none`).

```
<tbody>
<yfc:loopXML
binding="xml:/Order/ChargeTransactionDetails/@ChargeTransactionDetail"
id="ChargeTransactionDetail">
<%request.setAttribute("ChargeTransactionDetail",
(YFCElement)pageContext.getAttribute("ChargeTransactionDetail"));%>
        <yfc:makeXMLInput name="InvoiceKey">
            <yfc:makeXMLKey binding="xml:/GetOrderInvoiceDetails/@InvoiceKey"
value="xml:/ChargeTransactionDetail/@OrderInvoiceKey" />
        </yfc:makeXMLInput>
        <tr>
            <td class="tablecolumn"
sortValue="<%=getDateValue("xml:ChargeTransactionDetail:/ChargeTransactionDetail
/@Createts")%>">
                <yfc:getXMLValue
binding="xml:/ChargeTransactionDetail/@Createts"/>
            </td>
            <td class="tablecolumn">
                <yfc:getXMLValue
binding="xml:/ChargeTransactionDetail/@ChargeType"/>
                <% if
(equals("AUTHORIZATION",getValue("ChargeTransactionDetail","xml:/ChargeTransacti
onDetail/@ChargeType")) ||
equals("CHARGE",getValue("ChargeTransactionDetail","xml:/ChargeTransactionDetail
/@ChargeType"))) {%>
                    <% String divToDisplay="yfsPaymentInfo_" +
ChargeTransactionDetailCounter; %>
                    <img
onclick="expandCollapseDetails('<%=divToDisplay%>','<%=getI18N("Click_To_See_
Payment_Info")%>','<%=getI18N("Click_To_Hide_Payment_
Info")%>','<%=YFSUIBackendConsts.FOLDER_
COLLAPSE%>','<%=YFSUIBackendConsts.FOLDER_EXPAND%>')" style="cursor:hand"
<%=getImageOptions(YFSUIBackendConsts.FOLDER,"Click_To_See_Payment_Info")%>/>
                    <div id=<%=divToDisplay%>
style="display:none;padding-top:5px">
                        <table width="100%" class="view">
                            <tr>
                                <td height="100%">
                                    <jsp:include page="/om/Orderdetail/order_
```

```
detail_paymenttype_collections.jsp">
                                       <jsp:param name="PrePathId"
value="ChargeTransactionDetail"/>
                                       <jsp:param name="ShowAdditionalParams"
value="Y"/>
                                       <jsp:param
name="DecryptedCreditCardLink" value="L02"/>
                                     </jsp:include>
                                  </td>
                               </tr>
                            </table>
                         </div>
                   <%}%>
                </td>
                <td class="numerictablecolumn"
sortValue="<%=getNumericValue("xml:ChargeTransactionDetail:/ChargeTransactionDet
ail/@CreditAmount")%>">
                   <yfc:getXMLValue
binding="xml:/ChargeTransactionDetail/@CreditAmount"/>
                </td>
          </tr>
     </yfc:loopXML>
</tbody>
```

C.6.5 getAttributeNameFromBinding

This JavaScript function parses the binding string passed as input and
returns the attribute from the string.

Syntax

getAttributeNameFromBinding(str)

Input Parameters

str - Optional. String containing the binding string. If not passed, the
function returns null.

Return Value

Attribute portion of the binding string.

C.6.6 getCurrentSearchViewId

This JavaScript function retrieves the Resource ID of the current search view. This function can be used only for search views. To get the Resource ID of the current detail view, use the getCurrentViewId JavaScript function on the detail view JSP page. See "getCurrentViewId" on page 655.

Syntax

getCurrentSearchViewId()

Input Parameters

None.

Return Value

Resource ID of the current search view.

Example

This example shows how to refresh the current search view when a value is selected from a combo box by obtaining the current View ID.

```
<select class="combobox" onChange="changeSearchView(getCurrentSearchViewId())"
<%=getComboOptions(documentTypeBinding)%>>
    <yfc:loopOptions
binding="xml:CommonDocumentTypeList:/DocumentParamsList/@DocumentParams"
name="Description"
    value="DocumentType" selected="<%=selectedDocumentType%>"/>
</select>
```

C.6.7 getCurrentViewId

This JavaScript function retrieves the Resource ID of the current detail view.

Syntax

getCurrentViewId()

Input Parameters

None.

Return Value

Resource ID of the current detail view.

Example

This example shows how to refresh the current view by obtaining the current View ID.

```
<td class="detaillabel" ><yfc:i18n>Horizon_End_Date</yfc:i18n></td>
<td class="protectedtext" nowrap="true">
    <input type="text" class="dateinput" onkeydown="return checkKeyPress(event)"
<%=getTextOptions("xml:/InventoryInformation/Item/@EndDate","xml:/InventoryInfor
mation/Item/@EndDate","")%> />
    <img class="lookupicon"  onclick="invokeCalendar(this);return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON,"View_Calendar")%> />
    <input type="button" class="button" value="GO"
onclick="if(validateControlValues())changeDetailView(getCurrentViewId())"/>
</td>
```

C.6.8 getObjectByAttrName

This JavaScript function returns the object that is bound to the specified attribute.

Binding is achieved through the use of a JSP function such as getTextOptions or getComboOptions. For more information on binding JSP functions, see "yfsGetTextOptions" on page 634 or "yfsGetComboOptions" on page 629.

Once a field is bound, the name attribute of that field contains the binding XML path. This JavaScript function searches for all input and combo boxes and text areas within the specified HTML tag, and matches the attribute portion of the name attribute. The first match is returned.

The attribute portion is separated from the rest of the name attribute by the at (@) separator. For example, if the name is xml:/Order/@ChargeNameKey, the attribute portion is ChargeNameKey.

Syntax

getObjectByAttrName(obj, attributeName)

Input Parameters

obj - Required. Handle to the HTML object under which the search is to be conducted.

attributeName - Optional. Attribute name for search to be conducted under the object specified. If not passed, the function returns null.

Return Value

Handle to the object that is bound to the attribute specified. If no such object is found, null is returned.

Example

This example shows how to enable and disable the charge name field on line taxes, based on the checking of a checkbox.

```
function setAsPriceCharge(thisCheckbox) {
 var checkboxName=thisCheckbox.name
 var trNode=getParentObject(thisCheckbox, "TR");
 var sel=getObjectByAttrName(trNode, "ChargeNameKey");

 if (sel != null) {
  if (thisCheckbox.checked) {
   sel.disabled=true;
   sel.value="";
  } else {
   sel.disabled=false;
  }
 }
}
```

C.6.9 getParentObject

This JavaScript function gets the first occurrence of the tag specified in the HTML ancestry of the passed object.

Syntax

getParentObject(obj, tag)

Input Parameters

obj - Required. Handle to an object in HTML object hierarchy.

tag - Optional. String containing the name of the ancestor node used in a search. If not passed, the function returns null.

Return Values
The first occurrence of the tag specified in the HTML ancestry of the passed object.

Example
This example shows how to code a client-side deletion to execute when the user selects a Delete icon in a table row. In this example, `element` refers to the object that the user selects.

```
function deleteRow(element) {
var row=getParentObject(element, "TR");

oTable=getParentObject(row, "TABLE");

row.parentNode.deleteRow(row.rowIndex - 1);

fireRowsChanged(oTable);

    return false;
}
```

C.6.10 goToURL
This JavaScript function opens a specified URL in a new window.

Syntax
goToURL(URLInput)

Input Parameters
URLInputObj - Optional. Name of the input tag that contains the URL specified by the user. If not passed, the function just returns.

Return Value
None.

Example

This example shows how the `goToUrl()` function opens the order instruction screen in a new window.

```
<td>
    <input type="text"
<%=yfsGetTextOptions("xml:/Order/Instructions/Instruction_" + InstructionCounter
+ "/@InstructionURL",
"xml:/Instruction/@InstructionURL","xml:/Order/AllowedModifications")%>/>
    <input type="button" class="button" value="GO"
onclick="javascript:goToURL('xml:/Order/Instructions/Instruction_
<%=InstructionCounter%>/@InstructionURL');"/>
</td>
```

C.6.11 ignoreChangeNames

Whenever any detail view is posted, the Sterling Presentation Framework checks for data changed through the screen controls. For controls that have no changes, the name attribute is changed to "old" + [current name]. By doing this, the data in these controls does not make it to the APIs. This results in improved performance, since unchanged data does not need to be updated. However, some APIs are designed to work in replace mode. They take a complete snapshot of information (including unchanged part) and replace the all of it in the database. For such APIs, all data from the screen must be passed as input.

To achieve this, this function can be called in the `onload` event. This function sets a custom property in the `window` object. When the screen is posted, the Sterling Presentation Framework checks for this custom property. If the property is set, the automatic name changing does not happen.

The Sterling Presentation Framework helps the user remember to save data they have input. When a user has changed some data and begins to navigate away from a page, the Sterling Presentation Framework detects the changed data and prompts the user to save their work. This function does not change the behavior of this feature in any way. It simply makes sure that the name property of the controls that have no changes are retained. In this way, this function differs from `yfcDoNotPromptForChanges()`. See "yfcDoNotPromptForChanges" on page 674.

Syntax
ignoreChangeNames()

Input Parameters
None.

Return Value
None.

Example
The example attaches this function to the `onload` event.

```
<script language="javascript">
window.attachEvent("onload", IgnoreChangeNames);
</script>
```

C.6.12 invokeCalendar

This JavaScript function invokes the calendar lookup. This function assumes that the previous object to the one passed (in the DOM hierarchy of the HTML) is the one that must be populated with the date selected in the lookup.

Syntax
invokeCalendar(obj)

Input Parameters
obj - Required. Handle to the image object that was selected to invoke the calendar.

Output Parameters
None.

Example
This example shows how the Calendar Lookup is invoked from the Horizon End Date field in the Inventory detail view.

```
<td class="detaillabel" ><yfc:i18n>Horizon_End_Date</yfc:i18n></td>
<td class="protectedtext" nowrap="true">
```

```
        <input type="text" class="dateinput" onkeydown="return
checkKeyPress(event)"
<%=getTextOptions("xml:/InventoryInformation/Item/@EndDate","xml:/InventoryInfor
mation/Item/@EndDate","")%> />
        <img class="lookupicon"  onclick="invokeCalendar(this);return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON,"View_Calendar")%> />
        <input type="button" class="button" value="GO"
onclick="if(validateControlValues())changeDetailView(getCurrentViewId())"/>
    </td>
```

C.6.13 invokeTimeLookup

This JavaScript function invokes the time lookup. The function assumes
that the previous object to the one passed (in the DOM hierarchy of the
HTML) is the one that must be populated with the date selected in the
lookup.

Syntax
invokeTimeLookup(obj)

Input Parameters
obj - handle to the image object that was clicked to invoke the calendar.

Output Parameters
None.

Example
This example shows how the time lookup is used in a date and time
search criteria field.

```
<tr>
   <td nowrap="true">
     <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@FromExpectedShipmentDate_YFCDATE")%>/>
     <img class="lookupicon" name="search" onclick="invokeCalendar(this);return
false" <%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar") %> />
     <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@FromExpectedShipmentDate_YFCTIME")%>/>
     <img class="lookupicon" name="search"
onclick="invokeTimeLookup(this);return false"
<%=getImageOptions(YFSUIBackendConsts.TIME_LOOKUP_ICON, "Time_Lookup") %> />
   <yfc:i18n>To</yfc:i18n>
```

```
          </td>
        </tr>
        <tr>
          <td>
            <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@ToExpectedShipmentDate_YFCDATE")%>/>
            <img class="lookupicon" name="search" onclick="invokeCalendar(this);return
false" <%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON, "Calendar") %> />
            <input class="dateinput" type="text"
<%=getTextOptions("xml:/Shipment/@ToExpectedShipmentDate_YFCTIME")%>/>
            <img class="lookupicon" name="search"
onclick="invokeTimeLookup(this);return false"
            <%=getImageOptions(YFSUIBackendConsts.TIME_LOOKUP_ICON, "Time_Lookup") %>/>
          </td>
        </tr>
```

C.6.14 showDetailFor

This JavaScript function changes the current page to show the default
view of the current entity. The resulting screen opens in the same
browser window, not in a new window. You typically use the
showDetailFor() function to move from the list view to the detail view.
Then after the detail view opens, this function is not used, because
subsequent views are typically invoked as popup windows and this
function does not do that.

If you do choose to use this function in a detail screen, the following
behavior must be kept in mind. This function does a [get] and not a
[post]. Therefore, if you see the Next or Previous icons on your screen
and you use this function to switch to the default view, the icons are lost.
The icons disappear because the hidden input in the current page that
contain information regarding the Next or Previous views are lost when
this function does a [get].

In a list screen, this function is used in conjunction with the
yfc:makeXMLInput JSP tag. The makeXMLInput JSP tag prepares an XML
containing the key attributes. That XML must be passed to the default
detail view.

Syntax
showDetailFor(entityKey)

Input Parameters

entityKey - Required. String containing a URL-encoded XML that contains the key attributes required by the detail view.

Output Parameters

None.

Example

This example shows an Order list view that contains two columns: Order Number and Enterprise Code. Order Number is hyperlinked to open the default detail view of Order. Notice that `yfc:makeXMLInput` prepares an XML that is later used as the input parameter to the `showDetailFor()` function by using the `getParameter()` JSP function.

```
<table class="table" editable="false" width="100%" cellspacing="0">
    <thead>
        <tr>
            <td sortable="no" class="checkboxheader">
                <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
            </td>
            <td class="tablecolumnheader"><yfc:i18n>Order_#</yfc:i18n></td>
            <td class="tablecolumnheader"><yfc:i18n>Enterprise</yfc:i18n></td>
        </tr>
    </thead>
    <tbody>
        <yfc:loopXML binding="xml:/OrderList/@Order" id="Order">
            <tr>
                <yfc:makeXMLInput name="orderKey">
                    <yfc:makeXMLKey binding="xml:/Order/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey" />
                </yfc:makeXMLInput>
                <td class="checkboxcolumn">
                    <input type="checkbox" value='<%=getParameter("orderKey")%>'
name="EntityKey"/>
                </td>
                <td class="tablecolumn"><a
href="javascript:showDetailFor('<%=getParameter("orderKey")%>');">
                    <yfc:getXMLValue binding="xml:/Order/@OrderNo"/></a>
                </td>
                <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/Order/@EnterpriseCode"/></td>
            </tr>
```

```
          </yfc:loopXML>
      </tbody>
</table>
```

C.6.15 showDetailForViewGroupId

This JavaScript function changes the current page to show the default view of the given View Group ID (The View ID with the least Resource Sequence number is the default view for a particular View Group Id). The resulting screen opens in the same browser window. Use the `showDetailForViewGroupId()function` to move from the list view to the detail view. When the detail view opens, this function is not used, because subsequent views are invoked as popup windows and this function does not do that.

In the list screen, this function is used in conjunction with the `yfc:makeXMLInput` JSP tag. The `makeXMLInput` JSP tag creates an XML containing the key attributes. That XML must be passed to the default detail view.

Syntax
showDetailForViewGroupId (entityname, viewGroupId, entityKey, extraParameters)

Input Parameters
entityName - Required. Entity to search in the detail screen.

viewGroupId - Required. The view group ID shown to the user.

entityKey - Required. String containing a URL-encoded XML that contains key attributes required by the detail view.

Output Parameters
None.

Example
```
<td class="tablecolumn">
     <a href = "javascript:showDetailForViewGroupId
('load','YDMD200','<%=getParameter("loadKey")%>');">   <yfc:getXMLValue
binding="xml:/Load/@LoadNo"/>
     </a>
</td>
```

C.6.16 showHelp

This JavaScript function invokes online help in a new window.

Online help can be internationalized.

Syntax
showHelp()

Input Parameters
None.

Returns
None.

Examples
The following example shows how online help is invoked when the help icon is selected from the menu bar and it opens to the table of contents.

```
<img alt="<%=getI18N("Help")%>" src="<%=YFSUIBackendConsts.YANTRA_HELP%>"
onclick='showHelp();'/>
```

> **Note:** The screen-level Help is available only for system-defined search, list, and detail views. The functionality for custom views is provided through a different internal javascript function. This `showHelp` function is exposed mainly for use from the menu bar, which is customizable. From the menu, you will typically want only the overall system help and not the screen-level context sensitive help.

C.6.17 showPopupDetailFor

This JavaScript function shows the default view of the current entity in a popup window (modal dialog). It is a blocking call. It does not return until the modal dialog is closed.

Syntax
showPopupDetailFor(key, name, width, height, argument)

Input Parameters

key - Required. Entity key that is required by the detail view. If not passed, the current entity's key is automatically passed to the popup window.

name - Required. Pass as blank space (" "). Not used.

width - Required. Horizontal size of the popup window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels. If passed as 0, a certain default height is used.

argument - Required. Anything passed in this field is available in the modal dialog through the `window.dialogArguments` attribute.

Returns

None.

Example

This example shows how the inventory audit detail is invoked from the inventory audit list screen.

The same list screen is used in a list view, as well as in a detail popup window. When you select the transaction date, if the current screen is a popup window, another popup window is invoked with the audit details. If the current view is list view, the audit detail screen comes up in the same window.

```
<tbody>
    <yfc:loopXML name="InventoryAudits"
binding="xml:/InventoryAudits/@InventoryAudit" id="InventoryAudit">
    <tr>
        <yfc:makeXMLInput name="inventoryAuditKey">
            <yfc:makeXMLKey binding="xml:/InventoryAudit/@InventoryAuditKey"
value="xml:/InventoryAudit/@InventoryAuditKey" />
            <yfc:makeXMLKey binding="xml:/InventoryAudit/@OrganizationCode"
value="xml:/InventoryAudit/@InventoryOrganizationCode" />
        </yfc:makeXMLInput>
        <td class="checkboxcolumn">
            <input type="checkbox"
value='<%=getParameter("inventoryAuditKey")%>' name="EntityKey"/>
        </td>
        <td class="tablecolumn"
```

```
sortValue="<%=getDateValue("xml:/InventoryAudit/@Modifyts")%>">
              <%if ( "Y".equals(request.getParameter(YFCUIBackendConsts.YFC_IN_
POPUP)) ) {%>
              <a href=""
onClick="showPopupDetailFor('<%=getParameter("inventoryAuditKey")%>',
'','900','550',window.dialogArguments);return false;" >
                  <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@Modifyts"/>
              </a>
              <%} else {%>
                  <a
href="javascript:showDetailFor('<%=getParameter("inventoryAuditKey")%>');">
                      <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@Modifyts"/>
                  </a>
              <%}%>
          </td>
          <td class="tablecolumn">
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@ItemID"/>
          </td>
          <td class="tablecolumn">
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@ProductClass"/>
          </td>
          <td class="tablecolumn">
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@UnitOfMeasure"/>
          </td>
          <td class="tablecolumn">
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@TransactionType"/>
          </td>
          <td class="tablecolumn">
              <yfc:getXMLValue name="InventoryAudit"
binding="xml:/InventoryAudit/@ShipNode"/>
          </td>
      </tr>
      </yfc:loopXML>
</tbody>
```

C.6.18 validateControlValues

This JavaScript function checks for client-side validation errors. When the user enters invalid data in an input field, the Sterling Presentation

Framework flags the field as in error. When the user submits data in the page, this function should be called to make sure that invalid data is not posted.

Syntax
validateControlValues()

Input Parameters
None.

Return Values
true - No errors were found.

false - One or more errors were found.

Example
This example shows how to check for errors before you submit the current page.

```
<td class="detaillabel" ><yfc:i18n>Horizon_End_Date</yfc:i18n></td>
<td class="protectedtext" nowrap="true">
    <input type="text" class="dateinput" onkeydown="return checkKeyPress(event)"
<%=getTextOptions("xml:/InventoryInformation/Item/@EndDate","xml:/InventoryInfor
mation/Item/@EndDate","")%> />
    <img class="lookupicon"  onclick="invokeCalendar(this);return false"
<%=getImageOptions(YFSUIBackendConsts.DATE_LOOKUP_ICON,"View_Calendar")%> />
    <input type="button" class="button" value="GO"
onclick="if(validateControlValues())changeDetailView(getCurrentViewId())"/>
</td>
```

C.6.19 yfcAllowSingleSelection

Some operations can be performed on only one record at a time. However, the user interface typically permits multiple options to be checked before an operation is selected. Therefore, the operations that do not support multiple selections must themselves validate that not more than one record has been selected for processing. This function does that validation.

Syntax
yfcAllowSingleSelection(chkName)

Input Parameters

chkName - Optional. Name of the set of checkbox controls, one of which must be checked before an operation is performed. If the value is not passed or is blanks, it defaults to *EntityKey*.

Output Parameters

true - Zero or one record was selected.

false - More than one record was selected.

Examples

Receiving intransit updates can only be done one stop at a time. Therefore, the operation for receiving intransit updates is configured to first call the JavaScript function `yfcAllowSingleSelection()` and then to invoke the `receiveIntransitUpdates()` API.

This example performs an action if one, and only one, selection was made for checkboxes that have the name set to the value passed in the sKeyName variable.

```
function goToOrderLineSchedules(sSearchViewID, sKeyName, bPopup)
{
  if(yfcAllowSingleSelection(sKeyName))
  {
   …
  }
 }
```

C.6.20 yfcBodyOnLoad

This JavaScript function is called whenever any page is loaded. Typically, it is automatically called when the page is loaded. However, if your page must do something special on the `onload` event, you can call this function first and then call your own `window.onload()` function.

Syntax

yfcBodyOnLoad()

Input Parameters

None.

Return Value

None.

Example

This example shows how the onload event can be taken by your custom JSP rather than let the Sterling Presentation Framework take it.

```
function window.onload(){
    if (!yfcBodyOnLoad() && (!document.all('YFCDetailError'))) {
        return;
    }

//Do your special processing here
}
```

C.6.21 yfcChangeDetailView

This JavaScript function switches to a specific detail view. This function uses the POST function to switch the view.

Syntax

yfcChangeDetailView(viewID)

Input Parameters

viewID - Required. Resource ID of the detail view to which you wish to switch.

Return Values

None.

Example

This example shows how to use the yfcChangeDetailView() function in order charges and the taxes summary in order to refresh the page to the current view when a combobox value is changed.

```
<select name="chargeType" class="combobox"
onchange="yfcChangeDetailView(getCurrentViewId());">
    <option value="Overall" <%if (equals(chargeType,"Overall")) {%> selected
<%}%>><yfc:i18n>Ordered</yfc:i18n></option>
    <option value="Remaining" <%if (equals(chargeType,"Remaining")) {%> selected
<%}%>><yfc:i18n>Open</yfc:i18n></option>
```

```
     <option value="Invoiced" <%if (equals(chargeType,"Invoiced")) {%> selected
<%}%>><yfc:i18n>Invoiced</yfc:i18n></option>
</select>
```

C.6.22 yfcChangeListView

This JavaScript function switches the current view to a list view. The list view is expected to have a pre-determined filter criteria, since this function does not accept any additional filter criteria.

Syntax
function yfcChangeListView(entity, searchViewId,maxrecords)

Input Parameters
entity - Required. Entity to which the searchViewId belongs.

searchViewId - Required. Identifier of the search view to which you wish to switch.

maxrecords - Optional. Maximum number of records to display in the list view. To enhance performance, use this parameter. If this is not passed, it defaults to the value specified in the yfs.properties file.

Output Parameters
None.

Example
The home page shows a list of alerts, up to a certain number, that has been set as the maximum number to display. To see a complete list of all alerts, the user can select the More Alerts operation. This operation is configured to call the yfcChangeListView() JavaScript function.

C.6.23 yfcCheckAllToSingleAPI

Deprecated in Sterling Supply Chain Applications version 5 SP1. On list views that have a Check All checkbox in the column header row, this function makes a single API call that takes in the multiple selections. Attach this function to the onclick event of the Check All checkbox.

This function works in conjunction with the yfcMultiSelectToSingleAPI() JavaScript function, which was also

deprecated in Sterling Supply Chain Applications version 5 SP1. For more information, see or its replacement .

Syntax
yfcCheckAllToSingleAPI(checkAllObject)

Input Parameters
checkAllObject - Required. Handle to the Check All checkbox in the table header.

Return Values
None.

Example
This example shows how to call this function when the Check All checkbox is selected.

```
<td sortable="no" class="checkboxheader">
            <input type="checkbox" name="checkbox" value="checkbox"
onclick='yfcCheckAllToSingleAPI(this)'/>
</td>
```

C.6.24 yfcDisplayOnlySelectedLines

This JavaScript function is for situations when the user needs to select multiple records from a list in screen A and those records must be passed on to screen B. In screen B, the selected records are displayed, possibly with additional information for each record. In such cases, the logic is that the same set of APIs that were used to build screen A could be called to also build screen B, and on the client side, a filtration process limits the display to only those selected in screen A.

This function requires that each row in the table that is under consideration must have an attribute called `yfcSelectionKey` set to the URL encoded XML (formed using `yfc:makeXMLInput` JSP tag).

Syntax
yfcDisplayOnlySelectedLines(tableId)

Input Parameters

tableId - Required. Identifier attribute of the table whose content must be limited to that selected from the previous screen.

Output Parameters

None.

Example

The following example shows how the create order line dependency screen limits the results in the order lines list to the specific lines that are selected in the order detail screen. First, this function must be called in the onload event.

```
<script language="javascript">
    function window.onload() {
        if (!yfcBodyOnLoad() && (!document.all('YFCDetailError'))) {
            return;
        }
        yfcDisplayOnlySelectedLines("DependentLines");
    }
</script>
```

Second, each <tr> tag must contain the yfcSelectionKey attribute.

```
            <tbody>
                <yfc:loopXML name="Order"
binding="xml:/Order/OrderLines/@OrderLine" id="OrderLine">
                        <yfc:makeXMLInput name="orderLineKey">
                            <yfc:makeXMLKey
binding="xml:/OrderLineDetail/@OrderLineKey"
value="xml:/OrderLine/@OrderLineKey"/>
                            <yfc:makeXMLKey
binding="xml:/OrderLineDetail/@OrderHeaderKey"
value="xml:/Order/@OrderHeaderKey"/>
                        </yfc:makeXMLInput>
                    <tr yfcSelectionKey="<%=getParameter("orderLineKey")%>">
                        <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@ItemID"/></td>
                        <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@ProductClass"/></td>
                        <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@UnitOfMeasure"/></td>
                        <td class="tablecolumn"><yfc:getXMLValue
binding="xml:/OrderLine/Item/@ItemDesc"/></td>
```

```
                    </tr>
                </yfc:loopXML>
            </tbody>
        </table>
```

C.6.25 yfcDoNotPromptForChanges

This JavaScript function turns off the automatic prompts that remind the user to save changes to their data. By default on any screen, if a user enters data and then starts to navigate away without saving the data, the Sterling Presentation Framework catches this and alerts the user to save their data.

When you call this function it sets a parameter on the window object. This parameter is checked during the onunload event and if the parameter is set through this function, the user is not warned.

When you call this function to turn off prompting, all data in the screen is passed to the API during save.

This JavaScript function does not turn off the prompts that remind a user to save changes to their data when executing inner panel actions.

If you call an API on an inner panel and do not want the user to be prompted to save changes, you must also use either the yfcSetControlAsUnchanged or the yfcDoNotPromptForChangesForActions function. See yfcSetControlAsUnchanged on page 683 or yfcDoNotPromptForChangesForActions on page 675.

Syntax

yfcDoNotPromptForChanges(value)

Input Parameters

value - Required. Determines whether or not the user should be prompted to save any new data they have input that has not yet been saved. Valid values are true and false. If specified as true, the user is not prompted to save. If specified as false, the user is prompted to save the data.

Return Value

None.

Example

This example shows how this function turns off the automatic prompts for the manifest detail screen.

```
<script language="javascript">
yfcDoNotPromptForChanges(true);
</script>
```

C.6.26 yfcDoNotPromptForChangesForActions

This JavaScript function can be used when you want to skip the "Changes made to the current screen will be lost" validation that is done when the user clicks actions on an inner panel. Normally, inner panel actions in the Sterling Supply Chain Application Consoles are not used with the editable fields on a screen. Therefore, when the user changes an input field and clicks an action, the warning message will be displayed by default. Call this javascript method to avoid this validation. You can call this method for an all actions on a view by calling it in your JSP in a script tag. Alternatively, you can call this method for a specific action by calling it as part of the javascript property of the action resource.

Syntax

yfcDoNotPromptForChangesForActions(value)

Input Parameters

value - Required. Pass 'true' to skip the "changes made" validation. Pass 'false' to turn the validation on. By default, the validation is on.

Return Value

None.

Example

This example shows how to call the yfcDoNotPromptForChangesForActions function from a JSP to turn the "changes made" validation off for all actions on the view:

```
<script language="Javascript" >
   yfcDoNotPromptForChangesForActions(true);
</script>
```

C.6.27 yfcGetCurrentStyleSheet

This JavaScript function retrieves name of the style sheet for the current window.

Syntax

yfcGetCurrentStyleSheet()

Input Parameters

None.

Output Parameters

currentStyleSheet - The name of the style sheet for the current window. The full name of the style sheet is returned, including the file extension (for example, `sapphire.css`).

Example

This example shows how to get the current style sheet of the window.

```
var currentStyleSheet = yfcGetCurrentStyleSheet();
```

C.6.28 yfcGetSaveSearchHandle

This JavaScript function provides a handle to the Save Search icon on the search view. This handle then can be used for attaching events to achieve custom behavior. To change the behavior associated with the Search icon, see "yfcGetSearchHandle" on page 677.

Syntax

var oObj=yfcGetSaveSearchHandle();

Input Parameters

None.

Output Parameters

var - Handle to the Save Search icon on the search view.

Example

This example shows how to have the application perform custom processing when the user selects the Save Search icon.

```
<script language="javascript">
function attachBehaviorFn()
{
        ...
var oObj1=yfcGetSaveSearchHandle();
var sVal1=oObj1.attachEvent("onclick",fixDerivedFromReturnSearch);
}
    window.attachEvent("onload",attachBehaviourFn);
...
```

C.6.29 yfcGetSearchHandle

This JavaScript function provides a handle to the Search icon on a search view. This handle then can be used for attaching events in order to achieve custom behavior. To affect the behavior associated with the Save Search icon, see "yfcGetSaveSearchHandle" on page 676.

Syntax

var oObj=yfcGetSearchHandle();

Input Parameters

None.

Output Parameters

var - Handle to the Search icon on the search view.

Example

This example shows how to have the application perform custom processing when the user selects the Search icon.

```
<script language="javascript">
function attachBehaviourFn()
{
var oObj=yfcGetSearchHandle();
var sVal=oObj.attachEvent("onclick",fixDerivedFromReturnSearch);
        ...
}
    window.attachEvent("onload",attachBehaviourFn);
```

...

C.6.30 yfcHasControlChanged

This JavaScript function determines if the contents of a specific control have been modified by the user since the page loaded.

This is accomplished by comparing the current value of a specific control with the custom attribute `OldValue` stored in the control when the page is loaded.

In the case of checkboxes and radio buttons, the custom attribute is `oldchecked`.

Syntax

yfcHasControlChanged(ctrl)

Input Parameters

ctrl - Required. Object in the HTML object hierarchy.

Return Values

true - Value of the specified control is different from when the page was first loaded.

false - Value of the specified control is the same as when the page was first loaded.

Example

This example shows how the Order Modification Reasons popup window uses this function to set the Override Flag in a hidden field.

The hidden field is passed to the `changeOrder()` API only when a specific field (for example, requested ship date) that is permitted to be changed only by users with special override permissions is changed by the user. This function detects if any of the input in the screen has changed.

```
function setOverrideFlag(overrideFlagBinding) {

    var overrideFlagInput=document.all(overrideFlagBinding);

    var docInputs=document.getElementsByTagName("input");
    for (var i=0;i<docInputs.length;i++) {
```

```
            var docInput=docInputs.item(i);
            if (docInput.getAttribute("yfsoverride") == "true") {
                if (yfcHasControlChanged(docInput)) {
                    overrideFlagInput.value="Y";
                    return;
                }
            }
        }

        var docSelects=document.getElementsByTagName("select");
        for (var i=0;i<docSelects.length;i++) {
            var docSelect=docSelects.item(i);
            if (docSelect.getAttribute("yfsoverride") == "true") {
                if (yfcHasControlChanged(docSelect)) {
                    overrideFlagInput.value="Y";
                    return;
                }
            }
        }
    }
```

C.6.31 yfcMultiSelectToSingleAPI

Deprecated in Sterling Supply Chain Applications version 5 SP1. Replaced by "yfcMultiSelectToSingleAPIOnAction" on page 681.

This JavaScript function enables you to make a single API call using multiple selections in a list. By default, when an action that calls an API on a list screen is executed while multiple selections have been made by the user, the API is invoked once for each selected record. This function enables you to configure an action that calls and API that is executed only once for all selected records. This function should be attached to the onclick event of each of the selection checkboxes of a list screen. This function will create hidden inputs on the list screen for the record being selected. Assuming that the input namespace of the action has been defined correctly, the API is called and all selected records are passed.

Syntax
yfcMultiSelectToSingleAPI(checkboxObject, counter, keyAttributeName, keyAttributeValue, parentNodePrefix, parentNodePostfix)

Input Parameters

checkboxObject - Required. Handle to the checkbox object (in HTML object hierarchy) on a table header.

counter - Required. Counter that uniquely identifies the row containing the checkbox object passed in the first parameter.

keyAttributeName - Required. Name of the attribute to be passed to the API.

keyAttributeValue - Required. Value of the attribute name specified in the keyAttributeName parameter.

parentNodePrefix - Required. Portion of the XML binding, up to and including the repeating XML element, to be passed to the API.

parentNodePostfix - Optional. Portion of the XML binding between the repeating XML element of the API input up to the final element in which the attribute specified in the keyAttributeName parameter is to be passed. Use this parameter only when the attribute specified in the keyAttributeName parameter is located under a child XML element of the repeating XML element. If not passed, it is assumed that the attribute specified in the keyAttributeName parameter is directly under the repeating element.

Return Values

None.

Example

This example shows how an Add to Shipment action can be processed in an Order Release list view.

In this example, the JSP code loops through a list of order releases and places a checkbox in the first <td> tag of the row. The checkbox's `onclick` event calls the `yfcMultiSelectToSingleAPI()` JavaScript function, which creates the hidden inputs required for the Add to Shipment action.

Note that the `yfcMultiSelectToSingleAPI()` JavaScript function is called twice to create two hidden inputs required by the API.

```
<yfc:loopXML binding="xml:/OrderReleaseList/@OrderRelease" id="OrderRelease">
      <tr>
          <yfc:makeXMLInput name="orderReleaseKey">
```

```
                <yfc:makeXMLKey
binding="xml:/OrderReleaseDetail/@OrderReleaseKey"
value="xml:/OrderRelease/@OrderReleaseKey" />
                <yfc:makeXMLKey
binding="xml:/OrderReleaseDetail/@OrderHeaderKey"
value="xml:/OrderRelease/@OrderHeaderKey" />
                <yfc:makeXMLKey binding="xml:/OrderReleaseDetail/@ReleaseNo"
value="xml:/OrderRelease/@ReleaseNo" />
            </yfc:makeXMLInput>
            <td class="checkboxcolumn"><input type="checkbox"
value='<%=getParameter("orderReleaseKey")%>' name="EntityKey"
                onclick=' yfcMultiSelectToSingleAPI (this ,
"<%=OrderReleaseCounter%>", "OrderReleaseKey", "<%=getValue("OrderRelease",
"xml:/OrderRelease/@OrderReleaseKey")%>",
"xml:/Shipment/OrderReleases/OrderRelease", null);
                yfcMultiSelectToSingleAPI (this , "<%=OrderReleaseCounter%>",
"OrderHeaderKey", "<%=getValue("OrderRelease",
"xml:/OrderRelease/@OrderHeaderKey")%>",
"xml:/Shipment/OrderReleases/OrderRelease", null)/>
            </td>
            <…>
    </tr>
</yfc:loopXML>
```

C.6.32 yfcMultiSelectToSingleAPIOnAction

This JavaScript function replaces the `yfcMultiSelectToSingleAPI()`
JavaScript function, which was deprecated in Sterling Supply Chain
Applications version 5 SP1.

This function makes a single API call using multiple selections in a list. By
default, when an action that calls an API on a list screen is executed
while multiple selections have been made by the user, the API executes
once for each selected record. This function enables you to configure an
action that calls an API that executes only *once* for *all* selected records.

Attach this function to the action resource. This function creates hidden
inputs on the list screen for each record that the user selects. Assuming
that the input namespace of the action has been defined correctly, the
API is called once and all selected records are passed.

Syntax
yfcMultiSelectToSingleAPIOnAction(checkBoxName, counterAttrName,
valueAttrName, keyAttributeName, parentNodePrefix, parentNodePostfix)

Input Parameters

checkBoxName - Required. Name of the checkbox object within the JSP where the action is defined.

counterAttrName - Required. Name of the HTML attribute on the checkbox object that contains the counter value that uniquely identifies this row within the table. It is recommended that you always use the string `yfcMultiSelectCounter` for this parameter.

valueAttrName - Required. Name of the HTML attribute on the checkbox object that contains the value that should be set into the attribute passed in the keyAttributeName parameter. It is recommended that you prefix the value with the `yfcMultiSelectValue` string. For more details, see the example.

keyAttributeName - Required. Name of the attribute that should be passed to the API.

parentNodePrefix - Required. The portion of the XML binding up to and including the repeating XML element that is to be passed as input to the API.

parentNodePostfix - Optional. The portion of the XML binding between the repeating XML element of the API input up to the final element in which the attribute specified in the keyAttributeName parameter is to be passed.

Output Parameters

None.

Example

This example shows a Create Shipment action that has been defined on a Order Release list view. The following shows how the checkbox object is created within the JSP:

```
<td class="checkboxcolumn"><input type="checkbox"
value='<%=getParameter("orderReleaseKey")%>' name="EntityKey"
yfcMultiSelectCounter='<%=OrderReleaseCounter%>'
yfcMultiSelectValue1='<%=getValue("OrderRelease",
"xml:/OrderRelease/@OrderReleaseKey")%>'
yfcMultiSelectValue2='Add'/>
</td>
```

Additionally, the Create Shipment action has the JavaScript field set to the following:

```
yfcMultiSelectToSingleAPIOnAction('EntityKey', 'yfcMultiSelectCounter',
'yfcMultiSelectValue1', 'OrderReleaseKey',
'xml:/Shipment/OrderReleases/OrderRelease',
null);yfcMultiSelectToSingleAPIOnAction('EntityKey', 'yfcMultiSelectCounter',
'yfcMultiSelectValue2', 'AssociationAction',
'xml:/Shipment/OrderReleases/OrderRelease', null);
```

Note that the `yfcMultiSelectToSingleAPIOnAction` function is called *twice* to create two hidden inputs required by the API.

C.6.33 yfcSetControlAsUnchanged

This JavaScript function eliminates prompting the user to save data. when controls are placed on an inner panel. Achieves this by setting controls as "not changed." The something function sets the prompt "Changes made to the data on screen will be lost" from appearing. For more information on users' changes to controls, see "ignoreChangeNames" on page 659.

After configuring all controls on a page to use this function, call this function for each control on a page before invoking an action.

If an inner panel uses an Action and has modifiable controls that take input required for the Action, you can use this function to prevent the "Changes made to …" message.

When using this function, you must also call the `yfcDoNotPromptForChanges()` function in the JSP containing the Action. For more information about user prompts, see "yfcDoNotPromptForChanges" on page 674.

Syntax
yfcSetControlAsUnchanged (control)

Input Parameters
control - Required. Object in the HTML object hierarchy.

Return Value
None.

Example

This example shows how to call the `CallSetControl()` function from Action:

```
<script language="javascript"> yfcDoNotPromptForChanges(true) </script>
<script language="javascript">
function CallSetControl() {
 var myControl=document.all("xml:/InventoryItem/SKU/@OldSKU");
 var myControl_1=document.all("xml:/InventoryItem/SKU/@NewSKU");
 var myControl_2=document.all("xml:/InventoryItem/@EMailID");

 yfcSetControlAsUnchanged(myControl);
 yfcSetControlAsUnchanged(myControl_1);
 yfcSetControlAsUnchanged(myControl_2);
 return(true);
 }
 </script>
```

C.6.34 yfcShowDefaultDetailPopupForEntity

This JavaScript function shows the default detail view of an entity in a popup window (modal dialog). The entity for which the view is displayed must be specified in the yfsTargetEntity attribute of the checkbox object whose name is passed as input. It is a blocking call. It does not return until the modal dialog is closed.

Syntax

yfcShowDefaultDetailPopupForEntity(checkBoxName)

Input Parameters

checkBoxName – Required. Name of one or more checkbox objects with the yfsTargetEntity attribute containing the ID of the entity for which the default detail view is to be displayed.

Return Values

None.

Example

This example shows how a view details action on an Order List screen could use this function to bring up the default detail view of the order entity.

JSP code for the checkbox:

```
<td class="checkboxcolumn">
<input type="checkbox" value='<%=getParameter("orderKey")%>'
name="chkRelatedKey" yfsTargetEntity="order"/>
</td>
```

The view details action should be defined with these properties:

```
ID="<Some ID>"
Name="View_Details"
Javascript="showDefaultDetailPopupForEntity('chkRelatedKey')"
Selection Key Name="chkRelatedKey"
```

C.6.35 yfcShowDetailPopup

This JavaScript function shows a specific view ID in a popup window, which is modal. It is a blocking call; it does not return until the modal dialog box is closed.

Syntax

yfcShowDetailPopup(viewID, name, width, height, argument, entity, key)

Input Parameters

viewID - Required. Resource ID of the detail view to be shown as a popup window. If passed as an empty string, the popup window displays the default detail view of the entity specified in the entity parameter.

name - Required. Pass as blank space (" "). Not used.

width - Required. Horizontal size of the popup window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels. If passed as 0, a certain default height is used.

argument - Required. Anything passed in this field is available in the modal dialog through the `window.dialogArguments` attribute.

entity - Optional. The entity of the detail view that is to be opened. If not passed, defaults to the same entity of the view that is currently being displayed.

key - Optional. Entity key that is required by the detail view. If not passed, the key of the current entity is passed to popup window.

Return Values

None.

Example

This example shows how the Modification Reason Code popup window displays when Save is selected on the Order Detail screen.

```
function enterActionModificationReason(modReasonViewID, modReasonCodeBinding,
modReasonTextBinding) {

    var myObject=new Object();
    myObject.currentWindow=window;
    myObject.reasonCodeInput=document.all(modReasonCodeBinding);
    myObject.reasonTextInput=document.all(modReasonTextBinding);

    // If the current screen has a hidden input for draft order flag
    // and the value of the input is "Y", don't show the modification
    // reason window.
    var draftOrderInput=document.all("hiddenDraftOrderFlag");
    if (draftOrderInput != null) {
        if ("Y" == draftOrderInput.value) {
            return (true);
        }
    }

    yfcShowDetailPopup(modReasonViewID, "", "550", "255", myObject);

    if (getOKClickedAttribute() == "true") {
        window.document.documentElement.setAttribute("OKClicked", "false");
        return (true);
    }
    else {
        window.document.documentElement.setAttribute("OKClicked", "false");
        return (false);
    }
}
```

C.6.36 yfcShowDetailPopupWithDynamicKey

When called with a specific object (in the HTML object hierarchy this JavaScript function prepares a URL-encoded XML containing all the values under the object (recursively). Only the values to be posted are

considered. The XML then is passed on to a popup window as a parameter to show the specified view.

Syntax
yfcShowDetailPopupWithDynamicKey(obj, view, entity, inputNodeName, winObj)

Input Parameters
obj - Required. Handle to the object based on which the key is dynamically prepared. For the specific object, this function traverses up the HTML hierarchy to find the nearest <table> tag. From that <table> tag, this function then searches for all input and checkboxes. Based on the binding for these controls, a URL encoded XML is formed which is then passed as the entity key to the detail view being invoked.

viewID - Required. Resource ID of the detail view to be shown as a popup window. If passed as an empty string, the default detail view of the specified entity is shown in the popup window.

entity - Optional. Resource ID of the entity of the detail view to be shown. If not passed, defaults to the current entity.

inputNodeName - Required. Root node name of the XML to be prepared to be passed to the detail view.

winObj - Optional. Anything passed in this field becomes available in the modal dialog through the `window.dialogArguments` attribute. If this parameter is not passed, an empty object is passed to the popup window.

Return Value
None.

Example
This example shows how to use this function to invoke list of order lines that can be added to a return. The list of order lines requires an order number to be specified, but this number is editable by the user. Hence, the input cannot be formed on the server side through a makeXMLInput JSP tag. Therefore, the input is prepared on the client side using this function. The following example contains a `doClick()` function that must be configured to be called when you select a Proceed icon as `doClick(this);`. This way, the button object itself is passed as a

parameter to the `doClick()` function. For this to work, the button object must be in the same <table> tag that contains the order number input box.

```
function okClick(obj) {
 yfcShowDetailPopupWithDynamicKey(obj, 'YOMD2002', 'return', 'Order',new
Object());
}
```

C.6.37 yfcShowDetailPopupWithKeys

This JavaScript functions shows a specific view ID in a popup window (modal dialog). It is a blocking call. It does not return until the modal dialog is closed.

Use this function in situations where the default key generated by the Sterling Presentation Framework to be passed on the detail view is not accepted by the detail view being invoked.

Syntax

yfcShowDetailPopupWithKeys(viewID, name, width, height, argument, keyName, entity, selectionKeyName)

Input Parameters

viewID - Required. Resource ID of the detail view to be shown as a popup window. If passed as an empty string, the default detail view of the specified entity is displayed.

name - Required. Pass as blank space (" "). Not used.

width - Required. Horizontal size of the popup window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels. If passed as 0, a certain default height is used.

argument - Optional. Passed as the `argument` parameter to the `showModalDialog()` function that is used to show the popup window. This then becomes available in the modal dialog through the `window.dialogArguments` attribute. If not passed, a new Object is created and passed to the popup window.

keyName - Required. Name attribute of a control that contains the Entity Key that is required by the detail view. If it is not passed, defaults to the value EntityKey.

entity - Optional. Resource ID for the detail view being shown. If not passed, defaults to the current entity.

selectionKeyName - Optional. Name of the checkbox control that must be checked by the user before the popup window is invoked. If this name is not passed (or is passed as null), the check is not performed, and the popup window is invoked immediately.

Return Value
None.

Example
This example shows how to invoke the modify address dialog from an inner panel that specifies its own entity key.

```
function doModifyAddressDialogWithKeys(source, viewID, entityKeyName){
    var myObject=new Object();
    myObject.currentwindow=window;
    myObject.currentsource=source;

    if(viewID == null) {
        viewID="YADD001";
    }
    if (entityKeyName == null) {
        entityKeyName="EntityKey";
    }
    yfcShowDetailPopupWithKeys(viewID, "", "600", "425", myObject,
entityKeyName);
}
```

C.6.38 yfcShowDetailPopupWithParams

This JavaScript function invokes a specified detail view within a modal dialog. You can pass parameters to the detail view by forming a string in the format of name1=value1&name2=value2 and passing this string as a parameter to this function.

This function appends the passed string to the URL that is used to invoke the view. Thus, the passed parameters are available in the request object to the called view.

Syntax

yfcShowDetailPopupWithParams(viewID,name,width,height,params,entity
,key, argument)

Input Parameters

viewID - Required. Resource ID of the detail view to be shown as a
popup window. If passed as empty string, the default detail view of the
specified entity is displayed.

name - Required. Not used. However an empty string must be passed.

width - Required. Horizontal size of the popup window. Measured in
pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels.
If passed as 0, a certain default width is used.

params - Required. String containing parameters to be passed to the
detail view being invoked. Use the syntax
"name1=value1&name2=value2". This appends the string to the URL
invoking the detail view which enables the parameters to be available to
the detail view of the requested object.

entity - Optional. Resource ID corresponding the detail view. If not
passed, defaults to the current entity.

key - Optional. Value of the key to be passed as a parameter to the
detail view. If not passed, the current view's key is passed to the detail
view being invoked.

argument - Optional. Passed as the argument parameter to the
`showModalDialog()` function that is used to show the popup window.
This then becomes available in the modal dialog through the
`window.dialogArguments` attribute. If this is not passed, an empty
object is passed to the modal dialog.

Return Value

None.

Example

This example shows how the notes popup window is displayed using this
function. The *notes* popup window detail view requires certain
parameters to be passed to it. For instance, an XML binding pointing to

attributes that control if notes are editable for the current order status or not. To accomplish this, the following example forms a string containing these parameters and invokes this JavaScript function.

```
var
extraParams="allowedBinding=xml:/Order/AllowedModification&getBinding=xml:/Order
&saveBinding=xml:/Order";
yfcShowDetailPopupWithParams('YOMD020', '', "800", "600", extraParams);
```

C.6.39 yfcShowDetailPopupWithKeysAndParams

This JavaScript function invokes a specified detail view within a modal dialog. You can pass parameters to the detail view by forming a string in the format of `name1=value1&name2=value2` and passing this string as a parameter to this function.

This function appends the passed string to the URL that is used to invoke the view. Thus, the passed parameters are available in the request object to the called view.

Use this function in situations where the default key generated by the Sterling Presentation Framework to be passed on the detail view is not accepted by the detail view being invoked.

Syntax
yfcShowDetailPopupWithKeysAndParams(viewID, name, width, height, argument,keyName, entity, selectionKeyName, params)

Input Parameters
viewID - Required. Resource ID of the detail view to be shown as a popup window. If passed as an empty string, the default detail view of the specified entity is displayed.

name - Required. Pass as blank space (" "). Not used.

width - Required. Horizontal size of the popup window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels. If passed as 0, a certain default height is used.

argument - Optional. Passed as the `argument` parameter to the `showModalDialog()` function that is used to show the popup window. This then becomes available in the modal dialog through the

window.dialogArguments attribute. If not passed, a new Object is created and passed to the popup window.

keyName - Required. Name attribute of a control that contains the Entity Key that is required by the detail view. If it is not passed, defaults to the value EntityKey.

entity - Optional. Resource ID for the detail view being shown. If not passed, defaults to the current entity.

selectionKeyName - Optional. Name of the checkbox control that must be checked by the user before the popup window is invoked. If this name is not passed (or is passed as null), the check is not performed, and the popup window is invoked immediately.

params - Required. String containing parameters to be passed to the detail view being invoked. Use the syntax name1=value1&name2=value2. This appends the string to the URL invoking the detail view which enables the parameters to be available to the detail view of the requested object.

Return Value
None.

Example
This example opens a custom detail page and passes some custom parameters to it.

```
yfcShowDetailPopupWithKeysAndParams('CSTOrder012','',800,600,new
Object(),'EntityKey','order','EntityKey','CustParam1=xml:/Order&CustParam2=proce
ss')
```

C.6.40 yfcShowListPopupWithParams

This JavaScript function shows a specified list view in a popup window (modal dialog). This is a blocking call. The function does not return until the window is closed.

Syntax
yfcShowListPopupWithParams(viewID, name, width, height, argument, entity, params)

Input Parameters

viewID - Required. Resource ID of the list view to be shown as a popup window. If passed as an empty string, the default list view of the specified entity is displayed.

name - Required. Pass as a blank space (" "). Not used.

width - Required. Horizontal size of the popup window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels. If passed as 0, a certain default height is used.

argument - Required. Value passed as the `argument` parameter to `showModalDialog()` function that is used to show the popup window. This then becomes available in the modal dialog through the `window.dialogArguments` attribute.

entity - Optional. Resource ID for the detail view being shown. If not passed, defaults to the current entity.

params - Optional. String starting with an ampersand (&) and containing any extra parameters based on which the search is to be performed. The parameters passed become available to the list view being invoked as request parameters.

Return Value

None.

Example

This example shows how an inventory audit list is invoked directly from the Inventory Summary screen for a specified Organization, Item, UOM and Product Class.

```
function showInvAuditSearch(sViewID,sItemID,sUOM,sProductClass,sOrgCode)
{
      var ItemID=document.all(sItemID).value;
      var UOM=document.all(sUOM).value;
      var PC=document.all(sProductClass).value;
      var Org=document.all(sOrgCode).value;
      var entity="inventoryaudit";
      var
sAddnParams="&xml:/InventoryAudit/@ItemID="+ItemID+"&xml:/InventoryAudit/@UnitOf
Measure="+UOM;
      sAddnParams=sAddnParams +
```

```
"&xml:/InventoryAudit/@ProductClass="+PC+"&xml:/InventoryAudit/@OrganizationCode
="+Org;

        yfcShowListPopupWithParams(sViewID,"",'900', '500','',entity,
sAddnParams);
}
```

C.6.41 yfcShowSearchPopup

This function invokes the specified search view in a popup window. This function can be used to display lookup results.

Syntax

yfcShowSearchPopup(viewID, name, width, height, argument, entity)

Input Parameters

viewID - Required. Resource ID of the search view to be shown as a popup window. If passed as an empty string, the default detail view of the specified entity is displayed.

name - Required. Pass as a blank space (" "). Not used.

width - Required. Horizontal size of the popup window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the popup window. Measured in pixels. If passed as 0, a certain default height is used.

argument - Required. Value passed as the `argument` parameter to the `showModalDialog()` function that is used to show the popup window. This then becomes available in the modal dialog through the window.dialogArguments attribute.

entity - Optional. Resource ID corresponding to the entity being searched for. If not passed, defaults to the name of the current entity.

Return Value

None.

Example

This example shows how to invoke a single field lookup. The `callLookup()` function invokes a search popup window.

From the search popup window, when the user selects a row, the `setLookupValue()`function is called with the selected value as a parameter.

The `setLookupValue()` function populates the value in the text field and closes the lookup search window.

```
function setLookupValue(sVal)
{
    var Obj=window.dialogArguments
    if(Obj != null)
        Obj.field1.value=sVal;
    window.close();
}


//obj is to be passed as "this",
// which would be the icon that was selected for lookup.
//This function assumes that the lookup icon is placed
// immediately after the text field on which lookup is requested.
//entityname is the entity name of the search view
// that needs to be shown in the lookup.
function callLookup(obj,entityname)
{
var oObj=new Object();
var oField=obj.previousSibling;
while(oField != null && oField.type != "text" && oField.type != "TEXT")
{
oField=oField.previousSibling;
}
oObj.field1=oField;
yfcShowSearchPopup('','lookup',900,550,oObj,entityname);
}
```

C.6.42 yfcSpecialChangeNames

This JavaScript function must be called when an API requires that the entire row is passed if the key is passed.

Syntax
yfcSpecialChangeNames(id, checkOnlyBlankRow)

Input Parameters

id - Required. ID of the HTML tag under which the name changing must be performed.

checkOnlyBlankRow - Optional. If this is passed as `true`, only new blank rows (where all inputs and selects are void) are considered for changing names. If this is passed as `false`, then all the existing rows under the object whose ID is passed are considered. If not passed, the value defaults to `false`.

Return Value

None.

C.6.43 yfcSplitLine

This JavaScript function splits a specific row into two rows.

Syntax

yfcSplitLine(imageNode)

Input Parameters

imageNode - Required. Object pointer to the image that is selected in response to which this function is called.

Output Parameters

None.

Table C–5 lists the attributes to use at each cell level for determining the behavior of the newly created rows.

Table C–5   Cell Attributes

Attribute	Behavior
ShouldCopy	Determines whether or not to copy the contents of the cell, including child cells. If specified as true, the contents are copied. If specified as false, the contents are not copied and an empty cell is created. Defaults to false.
NewName	Determines whether a new cell acquires an automatically generated name. The generated name is derived from the name and row count of the current object being copied. If specified as true, the new cell acquires a new name. If specified as false, no name is generated. Defaults to false.
	The name generating logic requires that the original name contain an "_<integer>" at the spot where the row count must be inserted. For example, if the original name is `xml:/InspectOrder/ReceiptLines/FromReceiptLine_1/@ReceiptLineNo`, the new row has an object with the name as `xml:/InspectOrder/ReceiptLines/FromReceiptLine_2/@ReceiptLineNo`.
NewClassName	Class of the new copy of the object. For example, if the class of the original object is `unprotectedinput`, but if you want the new copy to be protected, specify the new class as protectedinput. If not used, the class of the original object is used in the copy.
NewDisabled	Determines whether or not controls are created in a disabled state. For some HTML controls (such as <img> tags), this means disabling all actions on the control. If specified as true, the property named `disabled` is set to true. If specified as false, the disabled property is not be set. Defaults to false.
NewResetValue	Determines the state of the value attribute for the new object. If this is set to true, the new object's value attribute is voided. For most HTML controls, this results in the contents of the control being blanked out. If specified as false, the value of the original control is set in the copy. Defaults to false.

Table C–5   Cell Attributes

Attribute	Behavior
NewContentEditable	Optional. Determines whether or not the new object inherits the ContentEditable property of the current object. If this is specified, the ContentEditable property of the current object also becomes the new object's ContentEditable property. For some HTML controls (such as text box), this property controls whether or not the control is editable. The value you specify becomes the value set in the ContentEditable attribute in the copy. If this is not specified, the new object inherits the current object's ContentEditable property.
NewTRClassName	Optional. Specify the class of a new row that is formed after a line split. If this attribute is not passed, the default behavior is seen.
	For example, if a TR had classname="oddrow" specified as the class and you want to retain the same class in the new row after a line split, then instead of <tr classname="oddrow" > the JSP should contain <tr classname="oddrow" NewTRClassName = "oddrow">.

Example

This example shows how you can split a line on the client side during returns inspection so that a specific receipt line can be given multiple dispositions.

```
<yfc:loopXML binding="xml:/ReceiptLines/@ReceiptLine" id="ReceiptLine">
<tr>
    <yfc:makeXMLInput name="receiptLineKey">
        <yfc:makeXMLKey binding="xml:/ReceiptLine/@ReceiptLineKey"
value="xml:/ReceiptLine/@ReceiptLineKey"/>
    </yfc:makeXMLInput>

    <td class="checkboxcolumn" ShouldCopy="false" nowrap="true">
        <input type="checkbox" value='<%=getParameter("receiptLineKey")%>'
name="chkEntityKey"/>
    </td>
    <td class="checkboxcolumn" nowrap="true" ShouldCopy="false" >
        <img class="columnicon" <%=getImageOptions(YFSUIBackendConsts.RECEIPT_
```

```
LINE_HISTORY, "Disposition_History")%>></a>
    </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="false" ><yfc:getXMLValue
binding="xml:/ReceiptLine/@SerialNo"/></td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="false" ><yfc:getXMLValue
binding="xml:/ReceiptLine/@LotNumber"/></td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="false" ><yfc:getXMLValue
binding="xml:/ReceiptLine/@ShipByDate"/></td>
    <td class="numerictablecolumn" nowrap="true" ShouldCopy="false"
><yfc:getXMLValue binding="xml:/ReceiptLine/@AvailableForTranQuantity"/></td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="false" ><yfc:getXMLValue
binding="xml:/ReceiptLine/@DispositionCode"/></td></td>
    <td class="tablecolumn" ShouldCopy="false" >
        <yfc:getXMLValue binding="xml:/ReceiptLine/@InspectionComments"/>
    </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="true" >
        <img IconName="addSplitLine" src="../console/icons/add.gif"
             <% if
(getNumericValue("xml:/ReceiptLine/@AvailableForTranQuantity") > 1) { %>
             class="lookupicon" onclick="yfcSplitLine(this)"
             <%} else {%>

style="filter:progid:DXImageTransform.Microsoft.BasicImage(grayScale=1)"
             <%}%>
             />
        <input type="hidden"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/@ReceiptHeaderKey",
"xml:/ReceiptLine/@ReceiptHeaderKey")%>/>
        <input type="hidden"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/@ReceiptLineNo", "xml:/ReceiptLine/@ReceiptLineNo")%>/>
        <select NewName="true" NewClassName="unprotectedinput"
NewContentEditable="true" NewResetValue="true"
           class="combobox"
<%=getComboOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@DispositionCode")%>>
        <yfc:loopOptions binding="xml:/ReturnDispositionList/@ReturnDisposition"
name="Description"
           value="DispositionCode"
selected="xml:/ReceiptLine/@DispositionCode"/>
        </select>
    </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="true" >
        <input type="text" NewName="true" NewClassName="numericunprotectedinput"
```

```
NewContentEditable="true" NewResetValue="true"
            class="numericunprotectedinput"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@Quantity", "")%>/>
    </td>
    <td class="tablecolumn" nowrap="true" ShouldCopy="true" >
        <input type="text" NewName="true" NewClassName="unprotectedinput"
NewContentEditable="true" NewResetValue="true"
            class="unprotectedinput"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@InspectionComments",
"")%>/>
    </td>
</tr>
</yfc:loopXML>
</tbody>
```

C.6.44 yfcValidateMandatoryNodes

This JavaScript function validates mandatory sections of a screen. The function parses through all the TABLE elements in a page, and for each of the tables, looks for the presence of the yfcMandatoryMessage attribute. If this attribute is found, the function looks into the contents of the table. If the contents have not changed, the function alerts the message set within the yfcMandatoryMessage attribute for the corresponding <table> tag.

Syntax
yfcValidateMandatoryNodes()

Input Parameters
None.

Output Parameters
None.

Example
The following example shows how mandatory validation is performed before Save in the case of return receiving. First, the Save Operation is configured to call the yfcValidateMandatoryNodes() function.

Second, in the inner panel JSP, the following attribute is set for the table that requires a validation check:

```
<table class="table" ID="ReceiveLines" width="100%" editable="true"
 yfcMandatoryMessage="<yfc:i18n>Receipt_information_must_be_entered</yfc:i18n>">
```

C.6.45 yfcFindErrorsOnPage

This JavaScript function can be used within JSPs. This function is used to find errors on a page. On finding an error, this function raises an appropriate alert message.

Syntax

yfcFindErrorsOnPage()

Input Parameters

None.

Return Value

None.

Example

This example shows how to call the yfcFindErrorsOnPage() function from a JSP. While adding a dynamic row inside a JSP, this function checks if the JSP has any errors on a page. On finding an error, an appropriate alert message is displayed:

```
function addRows(element) {

if(yfcFindErrorsOnPage())
return;
}
```

C.6.46 setRetrievedRecordCount

This JavaScript function can be used within JSPs created for list views. All list view screens typically have a message next to the title of the screen that indicates how many records were retrieved. For example, the message will display "Retrieved 2 record(s)" when the list view shows 2 records. You can use this javascript to set the count within this message dynamically. Typically, the UI infrastructure will automatically display the

correct message based on the output of the "List" API defined under the entity resource for this list view.

However, in some instances the "List" API cannot be used for a specific list view. In these cases, the list view as been set to ignore the default list API, and instead, calls its own API either by defining a different API under the list view resource or within its own JSP through the `callAPI` taglib. In this case, the UI infrastructure cannot automatically display the correct message.

Syntax
setRetrievedRecordCount (recordCount)

Input Parameters
recordCount - Required. The correct record count to display in the "Retrieved X record(s)" message.

Return Value
None.

Example
This example shows how to call the setRetrievedRecordCount() function from a JSP defined for a list view. The correct count is computed as JSP code. Then, this result is passed to the setRetrievedRecordCount method which is called inside a script tag:

```
<%
    YFCElement root = (YFCElement)request.getAttribute("OrganizationList");
    int countElem = countChildElements(root);
%>
<script language="javascript">
    setRetrievedRecordCount(<%=countElem%>);
</script>
```

C.7 Data Type Reference

The `DataType` node contains UIType and XMLType nodes. For the Sterling Presentation Framework, attributes specified in UIType override those specified in XMLType, which in turn override those specified in DataType.

Table C–6 lists which nodes are supported by specific datatype attributes.

Table C–6   Nodes Supporting DataType Attributes

Attribute	Description	Nodes Supported In
Name	Unique identifier of the abstract data type.	DataType Type
Type	Can take values NUMBER, VARCHAR2, DATE, DATETIME, QUANTITY. If Type is selected as QUANTITY, it may or may not take decimal values, based on the `yfs.install.displaydoublequantity=N` setting specified in the `yfs.properties` file.	DataType XMLType UIType
Size	If specified in the DataType node, it is taken as the maximum number of characters that can be entered in the input boxes. If specified the UIType node it is multiplied by 5, and the result is taken as the number of pixels to use for as the length of the input box.	DataType XMLType UIType
PpcSize	If specified in the DataType node, it is taken as the maximum number of characters that can be entered in the input boxes in the RF UI screens. If specified the UIType node it is multiplied by 5, and the result is taken as the number of pixels to use for as the length of the input box in the RF UI screens. In instances where the PpcSize attribute is not specified, the Size attribute is considered.	DataType XMLType UIType
ZeroAllowed	Used only for numeric fields.	DataType XMLType UIType

Table C–6   Nodes Supporting DataType Attributes

Attribute	Description	Nodes Supported In
UITableSize	The value specified here is multiplied by 5 and the result is used as the width in pixels. This specific attribute is available only upon special request. Use the `getUITableSize()` JSP function to eve this value.	UIType
NegativeAllowed	Used only for numeric fields.	DataType XMLType UIType

D

Mobile User Interface Extensibility Reference

Mobile device user interface extensibility is accomplished through scripts that determine how the user interface renders the screen and passes data. This chapter contains the following resources for you reference while developing the user interface:

- User Interface Style Reference

- Programming Standards

- JSP Functions

Also, as when doing any sort of development, see Appendix A, "Special Characters Reference".

D.1 User Interface Style Reference

The style reference helps you maintain consistency and enables easy code reusability.

D.1.1 Sterling Supply Chain Applications Mobile UI HTML Tags

The Sterling Supply Chain Applications mobile UI uses the following HTML tags:

Table D–1   Mobile Device Screen HTML Tags

Tag	Valid values/Detail
type	Valid values are: text, hidden, or button.
subtype	Text type tags use the following values: • Label - Static text. • ProtectedText - Non-editable input. • Text - Input text box. Hidden type tags use the value Hidden. Button type tags use the following values: • Command - HTML button. • CommandLogout - Logs the user out and displays the login prompt. • CommandBack - Switches to the previous view. • CommandNextView - Switches to the next view.
name	Name of the field. Cannot contain spaces.
value	Value of the field (Internationalized string from the resource bundle)
size	Length of the field.
maxlen	Maximum length of the field.
row	Row in which the field should appear.
col	Column in which the field should start.
validate	Input data validated by the server. Valid values are: • Always - validate input data in all cases. • True - validate input data only if the old value if different from the new value. • False - do not validate input data.

Table D–1   Mobile Device Screen HTML Tags

Tag	Valid values/Detail
mandatory	Field usage validated by the server. Valid values are: • True - the user is required to specify a value for the field. • False - the user is not required to specify a value for the field.
tag	Binding for the field recognized by infrastructure. The tag syntax is "binding=x", where 'x' is an XML binding (similar to the ones in Console screens).
url	Specified only for "button/Command" field type/subtype. If set, the request is forwarded to the value of this attribute. It should always be of the form `target + "?action=" + calledFormName` where target is the values of attribute "target" in the "form" element of the HMTL while calledFormName is the name of the JSP to be invoked without the ".jsp" extension.

D.1.2 JSP Tag Library

The Sterling Supply Chain Applications mobile UI screens use the same JSP tags listed in Appendix C, "Console JSP Interface Extensibility Reference".

D.1.3 JavaScript Functions

The Sterling Supply Chain Applications mobile UI screens use the JavaScript functions listed in the <YANTRA_ HOME>/Applications/Foundation/webpages/yfcscripts/mobile.js file.

The Sterling Supply Chain Applications mobile UI can use any of the Console functions if necessary. See in Appendix C, "Console JSP Interface Extensibility Reference".

D.1.4 Data Type Reference

The Sterling Supply Chain Applications mobile UI uses the same datatypes listed in Appendix C, "Console JSP Interface Extensibility Reference".

D.2 Programming Standards

Follow these programming standards to help you create and maintain files that are consistent, easy-to-read, and reusable.

D.2.1 Standards for Creating Well-Formed JSP Files

Although HTML code is embedded in Java Server Pages, strive to write JSP code that is easily readable. If you require some special XML manipulation that cannot be incorporated in the APIs, include a separate JSP file, so that HTML tags and Java code do not become mixed together.

Use the following standards when writing JSP files:

- Tab spacing - Set the editor tab spacing to 4.

- JavaScript files - Do not include any JavaScript in the JSP file. Put all JavaScript into a separate JS file.

- HTML tags - Type all HTML tags and attributes in lowercase letters.

- HTML attributes - Enclose all HTML element attribute values in double quotes. Single quotes and no quotes may work, but the standard is to use double quotes.

- HTML tables - Strictly adheres to XSD defined.

- Tags - Close all tags, whether required or not.

- Comments - Enclose all comments in the following manner:
  `<%/*……..*/%>`

> **Tip:** When finished coding a form, open it in any visual HTML editor to validate that the HTML is well-formed.

D.2.2 Internationalization

Sterling Presentation Framework enables you to write an internationalized application by providing the following features that can be customized to be *locale-specific*:

- i18n JSP tag for literals

- Server-side error messages

D.2.3 Validating Your HTML Files

Validate your HTML files. You can use any commercial software package or free, online application, such as the World Wide Web Consortium (W3C) HTML Validator at http://validator.w3.org/. As an alterative, when you finish coding a form, you can open it in any visual HTML editor to validate that the HTML is well-formed.

Additionally, validate the HTML files against the XSD files.

D.3 JSP Functions

Any of the following functions in the `<YANTRA_ HOME>/Applications/Foundation/webpages/yfc/rfutil.jspf` file can be included in any JSP for extending the Sterling Supply Chain Applications mobile UI. These functions are listed in alphabetical order.

You can use the following JSP functions:

D.3.1 addToTempQ

This mobile device JSP function adds the keyName and keyValue pair to TempQ in order to persistence data across JSPs. The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

This function also enables support of multiple duplicate key names.

Syntax

public void addToTempQ (String keyName, String keyValue, boolean allowDuplicates) throws Exception

public void addToTempQ (String keyName, String keyValue, Map m, boolean allowDuplicates) throws Exception

Input Parameters

keyName - Required. Name of the key to be stored in the TempQ.

keyValue - Required. Value of the key to be stored in the TempQ.

allowDuplicates - Required. Determines whether or not duplicate objects are allowed to be added to the TempQ.

m - Optional. Enables you to provide a map of name/value pairs in a java.util.map.

Example

This example shows how the addToTempQ function can be used when multiple cases have to be scanned and multiple CaseIDs have to be stored in a TempQ.

```
addToTempQ("Case", "Case", caseMap, true);
```

D.3.2 clearTempQ

This mobile device JSP function clears the TempQ. This is the first function to invoke before persisting any information in the TempQ. The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

Syntax

public void clearTempQ() throws Exception

Input Parameters

None.

Example

The following example shows how this function can be used to clear the TempQ:

```
clearTempQ();
```

D.3.3 deleteAllFromTempQ

This mobile device JSP function deletes entries from TempQ for a given keyName. Use this after an exception to clear CaseIds stored so far.

The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

Syntax

public void deleteAllFromTempQ(String keyName) throws Exception

Input Parameters

keyName - Required. This is the key for which TempQ entries will be deleted.

Example

This example shows how the deleteAllFromTempQ function can be used to remove all TempQ entries that correspond with the keyName CaseScanned.

```
deleteAllFromTempQ("CaseScanned");
```

D.3.4 deleteFromTempQ

This mobile device JSP function deletes the TempQ entry for a given keyName and keyValue pair.

The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

Syntax
public void deleteFromTempQ(String keyName, String keyValue) throws Exception

Input Parameters
keyName - Required. Name of the key in the TempQ that requires deletion.

keyValue - Required. Value of the key in the TempQ that requires deletion.

Example
This example shows how the `getLocale` function can be used in conjunction with the getDoubleFromLocalizedString function.

```
deleteFromTempQ("Case","Case");
```

D.3.5 getErrorXML

This mobile device JSP function returns an XML representation of error. The mobile device interprets this XML and renders an error page.

Syntax
public String getErrorXML(String error, String errorField)

public String getErrorXML(String error, String errorField, String severity)

Input Parameters
error - Required. Error description for the error to be shown to the user. This is usually derived by invoking the `checkForError()` function.

errorField - Required. Form field where the focus must be transferred to on clearing the error page.

severity - Optional. Displays the degree of error present. Recommended values in ascending order are: info, error, warning.

Example

The following example shows how this function can be used to get the error XML for rendering an error message on a mobile UI screen.

```
errorXML=getErrorXML(errorDesc, errorfield)
```

D.3.6 getField

This mobile device JSP function is used in conjunction with the `getForm()` function.

Syntax

public YFCElement getField(YFCDocument formDoc, String fieldName) throws Exception

Input Parameters

formDoc - Required. Name of the YFCDocument from which the element must be extracted.

fieldName - Required. Name of the form field for which the other attributes need to be set.

Example

This example shows the `getField` function.

Consider a form with formName as formName. The following code creates a YFCDocument from the XHTML form.

```
YFCDocument ydoc=getForm(formName);
```

Prior to setting the attributes of the form for a specific element, getField can be called as:

```
YFCElement dropoffLocationElem = getField(ydoc,"lblDropoffLocation");
```

To set the type and subtype attributes for the dropoffLocationElem, use:

```
dropoffLocationElem.setAttribute("type","hidden");
dropoffLocationElem.setAttribute("subtype","Hidden");
```

D.3.7 getForm

This mobile device JSP function reads the XHTML form for a given form name and returns a YFCDocument. The currentEntity name is prefixed to the formname and .html is suffixed. It looks for the file in the `<YANTRA_HOME>/Applications/Foundation/template/mobilescreens/` directory.

The `getForm()` function is always used in conjunction with the `getField()` function.

Syntax

public YFCDocument getForm(String formName) throws Exception

Input Parameters

formName - Required. Name of the XHTML form.

Example

The following example shows how this function can be used to return a YFCDocument for the form "formName".

```
YFCDocument ydoc=getForm()
```

D.3.8 getStoredElement

This mobile device JSP function returns the XML Element for all TempQ entries. Each keyName and keyValue entry can be obtained by traversing the Element.

Syntax

private YFCElement getStoredElement(YFCDocument ydoc, String keyName, String keyValue) throws Exception

Input Parameters

ydoc - Required. YFCDocument representation of the XHTML form.

keyName - Required. Name of the key in the TempQ.

keyValue - Required. Value of the key in the TempQ.

Example

The following example shows the `getStoredElement` function.

```
YFCElement criteria = getStoredElement(getTempQ(),"Criteria", "criteria");
```

D.3.9 getTempQ

This mobile device JSP function retrieves the TempQ document object from Session. If a TempQ document does not exist, this function creates one. Since the return type of this function is YFCDocument, it is used to get a handle to the TempQ and thereafter is used to get its elements or attributes for some of its elements.

The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

Syntax
public YFCDocument getTempQ() throws Exception

Input Parameters
None.

Example
The following example shows how this function can be used to first get the TempQ documents and then later to get the node list for elements with the tag name "LPN".

```
YFCNodeList lpnlist=()((getTempQ()), getElementsByTagName("LPN"));
```

D.3.10 getTempQValue

This mobile device JSP function returns the value of the TempQ for a specific key. The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

Syntax
private String getTempQValue(String keyName) throws Exception

Input Parameters
keyName - Required. Name of the key for which the TempQ value is to be returned.

Example

This example shows how the `getTempQValue` function can be used to return the keyValue corresponding to the `CaseScanned` key from the TempQ.

```
getTempQValue("CaseScanned");
```

D.3.11 replaceInTempQ

This mobile device JSP function replaces the value in TempQ for a given key. The TempQ utilities store name/value pair information on one page in the session and provide methods for accessing them on the subsequent screens.

Syntax

public void replaceInTempQ(String keyName, String keyValue) throws Exception

public void replaceInTempQ(String keyName, String keyValue, String newKeyValue) throws Exception

public void replaceInTempQ(String keyName, String keyValue, Map m) throws Exception

Input Parameters

keyName - Required. Name of the key in the TempQ that is being replaced.

keyValue - Required. Value of the key in the TempQ that is being replaced.

newKeyValue - Optional. The new value of the key in TempQ that replaces the old value.

m - Optional. Map that should replace the existing keyValue.

Example

The following example shows how this function can be used to change the value of RecordCount from "1" to "resultMap".

```
replaceInTempQ("RecordCountResult","1",resultMap);
```

D.3.12 resetAttribute

This mobile device JSP function removes the named attribute from request and PageContext.

> **Note:** It is a good coding practice to reset an attribute before using it in the code.

Syntax

public void resetAttribute(String name)

public void resetAttribute(String name, Object value)

Input Parameters

name - Required. The name of the request attribute that needs to be reset.

value - Optional. The value of the request attribute that the attribute should be reset to.

Example

This example shows how the `restAttribute` JSP function removes the named attribute from request and PageContext.

```
resetAttribute("TaskList","");
```

D.3.13 sendForm

This mobile device JSP function posts an HTML form and provides focus on a specific field on a subsequent JSP form. Three versions of syntax enable you to customize how data should display.

Syntax

public String sendForm(String formName, String focusField) throws Exception

public String sendForm(String formName, String focusField, boolean sendData) throws Exception

public String sendForm(YFCDocument formDoc, String focusField, boolean sendData) throws Exception

Input Parameters

formDoc - Required. Either formName of formDoc must be provided.

formName - Required. Either formName of formDoc must be provided.

focusField - Required. Form field where the focus must be transferred to in the invoked JSP.

sendData - Optional. Valid values: true and false.

Example

This example shows how the sendForm function can be used so that the form corresponding to the YFCDocument ydoc is posted. On invocation of the subsequent JSP, the focus is transferred to the txtLocationId field and data is posted.

```
sendForm(ydoc, "txtLocationId", true)
```

Index

J

L

M

N

O

Z