Sterling Selling and Fulfillment Foundation

IBM

Customizing Console JSP Interface for End-User

Version 9.1

Sterling Selling and Fulfillment Foundation

Customizing Console JSP Interface for End-User

Version 9.1

Note

Before using this information and the product it supports, read the information in "Notices" on page 179.

Contents

Chapter 1. Checklist for Customization Projects

Customization Projects

Projects to customize or extend Sterling Selling and Fulfillment Foundation vary with the type of changes that are needed. However, most projects involve an interconnected series of changes that are best carried out in a particular order. The checklist identifies the most common order of customization tasks and indicates which guide in the documentation set provides details about each stage.

The items identified for extension and/or modification in the documentation are Source Components (to the extent such item involves source code) and Sample Materials for purposes of the License Information file associated with this product.

Prepare Your Development Environment

Set up a development environment that mirrors your production environment, including whether you deploy your application on a WebLogic, WebSphere®, or JBoss application server. Doing so ensures that you can test your extensions in a real-time environment.

You install and deploy your application in your development environment following the same steps that you used to install and deploy it in your production environment. Refer to your system requirements and installation documentation for details.

You have an option to customize your application with Microsoft COM+. Using Microsoft COM+ has advantages such as increased security, better performance, increased manageability of server applications, and support for clients of mixed environments. If this is your choice, see the *Customization Basics Guide* about additional installation instructions.

Plan Your Customizations

Are you adding a new menu entry? Or customizing the sign-in screen or logo? Or customizing views or wizards? Or creating new themes or new screens? Each type of customization varies in scope and complexity.

For background, see the *Customization Basics Guide*, which summarizes the types of changes that you can make and provides important guidelines about file names, keywords, and other general conventions.

Extend the Database

For many customization projects, the first task is to extend the database so that it supports the other UI or API changes that you make later. For instructions, see the *Extending the Database Guide*, which includes information about the following topics:

- Important guidelines about what you can and cannot change in the database.

- Information about modifying APIs. If you modify database tables so that any APIs are impacted, you must extend the templates of those APIs or you cannot store or retrieve data from the database. This step is required if table modifications impact an API.
- How to generate audit references so that you improve record management by tracking records at the entity level. This step is optional.

Make Other Changes to APIs

Your application can call or invoke standard APIs or custom APIs. For background about APIs and the services architecture of service types, behavior, and security, see the *Customizing APIs Guide*. This guide includes information about the following types of changes:

- Invoke standard APIs for displaying data in the UI and for saving changes made in the UI to the database.
- Invoke customized APIs for executing your custom logic in the extended service definitions and pipeline configurations.
- APIs use input and output XML to store and retrieve data from the database. If you don't extend these API input and output XML files, you may not get the results you want in the UI when your business logic is executing.
- Every API input and output XML file has a DTD and XSD associated to it. Whenever you modify input and output XML, you must generate the corresponding DTD and XSD to ensure data integrity. If you don't generate the DTD and XSD for extended XMLs, you may get inconsistent data.

Customize the UI

IBM® applications support several UI frameworks. Depending on your application and the customizations you want to make, you may work in only one or in several of these frameworks. Each framework has its own process for customizing components such as menu items, logos, themes, and so on.

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

Depending on the framework you want, consult one of the following guides:

- *Customizing the Console JSP Interface Guide*
- *Customizing the Swing Interface Guide*
- *Customizing User Interfaces for Mobile Devices Guide*
- *Customizing the Rich Client Platform Guide* and *Using the RCP Extensibility Tool Guide*
- *Customizing the Web UI Framework Guide*

Extend Transactions

You can extend and enhance the standard functionality of your application by extending the Condition Builder and by integrating with external systems. For background about transaction types, security, dynamic variables, and extending the Condition Builder, see the *Extending Transactions Guide* and *Extending the Condition Builder Guide*. These guides includes information about the following types of changes:

- Extend the Condition Builder to define complex and dynamic conditions for executing your custom business logic and using a static set of attributes.
- Define variables to dynamically configure properties belonging to actions, agents, and services configurations.
- Set up transactional data security for controlling who has access to what data, how much they can see, and what they can do with it.
- Create custom time-triggered transactions. You can invoke and schedule custom time-triggered transactions in much the same manner as you invoke and schedule the time-triggered transactions supplied by your application.
- Coordinate your custom, time-triggered transactions with external transactions and run them either by raising an event, calling a user exit, or invoking a custom API or service.

Build and Deploy your Customizations or Extensions

After performing the customizations that you want, you must build and deploy your customizations or extensions.

1. Build and deploy your customizations or extensions in the test environment so you can verify them.
2. When you are ready, repeat the same process to build and deploy your customizations and extensions in your production environment.

For instructions about this process, see the *Customization Basics Guide* which includes information about the following topics:

- Building and deploying standard resources, database extensions, and other extensions (such as templates, user exits, and Java interfaces).
- Building and deploying enterprise-level extensions.

Chapter 2. Before You Customize the JSP Console

About Console JSP User Interface

The Presentation Framework enables you to change the way information is rendered (or displayed) in the Application Console, without changing the way it functions. Customizing the user interface requires writing scripts that determine how the user interface renders the screen and passes data.

This section describes how to customize the console user interface to suit your business needs. Each task is accomplished through a combination of configuration through the Applications Manager 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 Applications Manager, see the Sterling Selling and Fulfillment Foundation: *Configuration Guide* . For information on functions used within the JSP files, see "JSP Functions for the Console JSP Interface."

Important: When customizing the interface, copy the standard resources of your application and then modify your copy. Do not modify the standard resources for your application.

The following applications are shipped with Sterling Selling and Fulfillment Foundation:

- Sterling Selling and Fulfillment Suite: Application Console - The standard UI for creating, tracking, and viewing orders, item inventory, and returns .
- Applications Manager- The UI for configuring Sterling Selling and Fulfillment Foundation.

Guidelines for Customizing the Console JSP User Interface

Before you begin user interface extensibility, familiarize yourself with the guidelines listed in this section in order to help ensure a successful experience.

The HSDE screens are not extensible.

Compare JSPs

As part of the Sterling Selling and Fulfillment Foundation extensibility, you can extend JSP files of the current release.

When Sterling Selling and Fulfillment Foundation 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:

- The original JSP shipped with your original product version
- The extended JSP for the same original JSP
- The same JSP shipped as part of the new release

In order to facilitate the reconciliation process, IBM recommends that you consider purchasing a difference tool capable of performing a three-way comparison. For example, an inexpensive tool is Araxis. Please note that IBM does not resell this difference tool, nor do we warranty or guarantee it. It is merely provided as an example of an inexpensive tool. You are advised to do your own research on three-way comparison tools and choose the one appropriate for your 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 Applications Manager and then change the copy.
- Do not change any of the JSP files, JavaScript files, or icon JAR files supplied with the application. 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 your application. But in future releases, IBM may add more resources to these JSPs, which means you must monitor software changes and update your configuration to account for the 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 the two APIs with the necessary fields in the output template. Then, if IBM adds fields to the JSP file in future releases, you may need to modify your configuration to account for the 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 "Screens in the JSP Console Interface."

Prepare Your Development Environment

To start the customization process, you must prepare the development environment to accommodate for your changes.

Understand How to Find Reference Materials

The 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 Javadocs
- JavaScript - see "JavaScript Functions"
- JSP - see "JSP Functions for the Console JSP Interface"
- JSP Tags - see "JSP Tag Library for the Console JSP Interface"

Chapter 3. Request Handling and Themes in the JSP Console

About Request Handling and Themes in the JSP Console

In anySterling Selling and Fulfillment Foundation user interface that involves HTML forms, there is a possibility of duplicate requests being 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 potential problems created by duplicate requests, the application infrastructure framework uses page tokens to verify that only the original request is processed. When a user logs into the Application Console, a unique identifier called a 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. This safeguard is built into the application infrastructure framework.

About Centralized Themes

When a user successfully signs in, the standard application Home Page is opened. The look and feel of the Home Page is determined by the theme associated with the user's ID. The theme determines the fonts and colors to use in rendering graphical user interface elements such as screens, labels, and table headers.

supplies 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 Console and Application Manager. Ideally when defining themes, you should create the same theme for both applications in order to ensure a consistent user experience.

Application	Required Theme Files
Application Console	• 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.
Applications Manager	• XML - throughout

Do not modify any of the standard themes. Create your own CSS and XML files. In order to determine the classes used to define controls for each style, see "Controls and Classes."

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

Creating New Themes

About this task

Procedure

1. Copy the *INSTALL_DIR*/repository/xapi/template/merged/resource/sapphire.xml theme file to *INSTALL_DIR*/extensions/global/template/resource/*theme*.xml

   If the /extensions/global/template/resource/ directory structure does not exist, create the required directory structure.
2. Edit your new XML file to define the values you want to use for colors and fonts for each Color Name and Font Name element you want to display within the Applications Manager 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 *INSTALL_DIR*/repository/eardata/platform/war/css/sapphire.css theme style sheet to *INSTALL_DIR*/repository/eardata/platform/war/css/*theme*.css.
4. (Optional) Only if you have enabled skin, copy the *INSTALL_DIR*/repository/eardata/platform/war/skins/*skin-name*/css/*skin-name*_sapphire_mb.css theme style sheet to *INSTALL_DIR*/repository/eardata/platform/war/skins/*skin-name*/css/*skin-name_theme*_mb.css.
5. Edit your new style file to specify the new colors and fonts added in the new theme file.
6. Copy the *INSTALL_DIR*/repository/xapi/template/merged/resource/sapphire_exui.xml file to *INSTALL_DIR*/extensions/global/template/resource/*theme*_exui.xml.
7. Modify your new XML file to define the values you want to use for the colors and fonts for each Color Name and Font Name elements that you want to display within the Applications Manager and on any bar or pie charts.

   Note: Be aware that modifying fonts may require you to make other changes such as the size of fields or other UI elements that use the font. For more information about modifying fonts, see "CSS Theme File Reference."
8. From the Applications Manager menu, configure a theme.

   After creating a new theme, the user can select the appropriate theme from the drop-down list on the Application Console user interface. For information about defining themes, see the Sterling Selling and Fulfillment Foundation: *User Guide* .

Creating Themes for a User or Organization

About this task

The application allows you to define the UI style (logo, fonts, colors, and so forth) for a specific organization or user. You can also show different images based on different themes.

To define user-specific or organization-specific UI styles, add the theme-specific image jar in the *INSTALL_DIR*/repository/eardata/*application*/war. This theme specific image jar can also be localized. For example, *theme_name_locale*.jar.

When fetching images from the server, the application first looks for images in the theme-specific image jar. If the theme-specific jar is not present, or if the jar does not have a particular image, the framework searches for the particular image in the locale-specific jars in the following order:

Note: The theme-specific image jar overrides the other image jars except the *INSTALL_DIR*/repository/eardata/*application*/extn/yantraiconsbe.jar.

Procedure

1. *INSTALL_DIR*/repository/eardata/*application*/extn/yantraiconsbe.jar
2. *INSTALL_DIR*/repository/eardata/config/war/yfscommon/yantraiconsbe.jar
3. *INSTALL_DIR*/repository/eardata/*application*/*module_name*/*module-specific_icons_jar*

   Note: To display your theme-specific image in place of an existing image, make sure that the new theme-specific image you add to the theme-specific image jar has the same name and path as the existing one.

Chapter 4. Creating and Enabling New Plug-in Skins in the JSP Console

Support for Skins

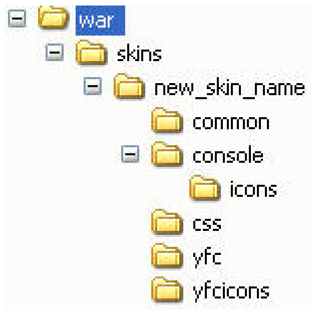
The application provides support for skins in HTML UI. Skins provide a better look and feel to the UI. In order to use skins, you must include the skin related UI files instead of default files.

Creating New Skins

You must store all the skin related files with the skin name under the *INSTALL_DIR*/repository/eardata/*application_name*/war/skins directory. A skin must have its own container, inner panel, menu bar, anchor JSPs and CSS files for all themes.

Directory Structure for Skins

When creating a new skin you should follow the following directory structure:



- common—This directory should contain the skin-specific menubar_anchor.jsp file and menubar_dropdown_multilevel.jsp file.

  The menubar_dropdown_multilevel.jsp file is required if you want to display submenus for a given menu item.
- console/icons—This directory should contain the skin-specific images. You can also store your localized images in this directory.

  Note: If you want to override the default images provided in the /console/icons/ and /yfcicons directories, you can copy the new images with the same name in the /skins/*skin-name*/console/icons and/or /skins/*skin-name*/yfcicons directories.
- css—This directory should contain your skin-specific css files. These css files are localizable. For example, *skin_name*_sapphire_mb.css, *skin-name*_jade_mb.css, *skin-name*_earth_mb.css, and so forth.
- yfc—This directory should contain skin-specific container_mb.jsp and innerpanel_mb.jsp files.
- yfcicons—This directory should contain your localization images.

  If the /yfcicons directory structure does not exist, create it.

Plug-in Points for Skins

From HTML UI point of view, the application provides four plug-in points where the skin-specific files should be included:

- /yfc/container_mb.jsp

  The skin-specific container JSP file should handle the search-list alignments which includes the list, or search, or detail JSP files and breadcrumbs JSP file. For example, a skin-specific container JSP file will look like this:

  ```
  <jsp:include page="/skins/skin-name/yfc/search_mb.jsp flush="true"/>
  <jsp:include page="/skins/skin-name/yfc/list_mb.jsp flush="true"/>
  <jsp:include page="/skins/skin-name/yfc/detail_mb.jsp" flush="true"/>
  ```

  The skin-specific search_mb.jsp, or list_mb.jsp, or yfc/detail_mb.jsp files should include the skin related UI file and framework provided functional files. For example, a skin-specific search JSP file will look like this:

  ```
  <jsp:include page="/yfc/search_functional_pre_mb.jsp" flush="true"/>>
  <jsp:include page="/skin-name/search_ui_mb.jsp" flush="true"/>
  <jsp:include page="/yfc/search_functional_post_mb.jsp" flush="true"/>
  ```

  When the skin is enabled, the framework provided /yfc/container_mb.jsp file will include the skin-specific /skins/*skin-name*/yfc/container_mb.jsp file.

- /yfc/innerpanel_mb.jsp

  The skin-specific inner panel JSP file should include the skin related UI file and framework provided functional files. For example, a skin-specific search JSP file will look like this:

  ```
  <jsp:include page="/yfc/innerpanel_functional_pre_mb.jsp" flush="true"/>
  <jsp:include page="/skins/skin-name/innerpanel_ui_mb.jsp" flush="true"/>
  <jsp:include page="/yfc/innerpanel_functional_post_mb.jsp" flush="true"/>
  ```

  When the skin is enabled, the framework provided /yfc/innerpanel_mb.jsp file will include the skin-specific /skins/*skin-name*/yfc/innerpanel_mb.jsp file

- /common/menubar_mb.jsp

  The skin-specific menu bar JSP files should define the layout and rendering of the menu bar. The framework provided menu bar file can also be included in this file (if required).

  Note: The skin-specific menubar_anchor.jsp and menubar_dropdown_multilevel.jsp files are used only if the framework provided common/menubar_mb.jsp file is used and no explicit context parameter is provided for menubar_anchor.jsp file.

  When the skin is enabled and no explicit context parameter is provided for the menu bar anchor JSP file, the framework-provided /common/menubar_mb.jsp file will include skin-specific /skins/*skin-name*/common/menubar_anchor.jsp file.

- CSS Files

  The skin-specific theme css files should refer to images as '../../../console/icons/*img-name*' or '../../../yfcicons/*img-name*' in order to support extensibility and localization. For example,

  ```
  .companyLogo{
  float:right;
  background: url(../../../console/icons/logo.jpg) no-repeat;
  height:42px;
  width:151px;
  margin-right:0;
  cursor:pointer;
    }
  ```

When the skin is enabled, the framework provided /yfc/container_mb.jsp file will include the skin-specific localized css files such as /skins/*skin-name*/css/*skin-name_theme-name*_mb.css file.

Note: By default, the original css files such as /css/*theme-name*.css will be included before the skin-specific css files.

Enabling New Skins

About this task

To enable a new skin, you must add the ‘yfc-ui-skin' context parameter in the *INSTALL_DIR*/repository/eardata/platform/descriptors/weblogic/WAR/WEB-INF/web.xml file. The name of the new skin should be specified as value of this parameter. For example, the entry in the web.xml file will look like this:

```
<context-param>
<param-name>yfc-ui-skin</param-name>
<param-value>skin_name</param-value>
</context-param>
```

Chapter 5. Customizing the Sign In Screen for the JSP Console

About the Sign In Screen

The Sign In screen is the first page that displays when you start the application. The Sign In screen comprises 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 files open in the order shown in the following figure.

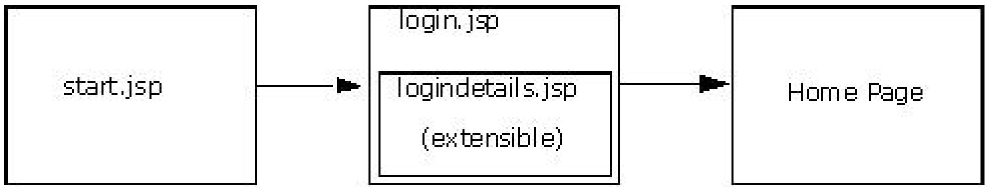


Figure 1. Sign In Screen Logic

The primary purpose of the Sign In screen is to enable authorized users to log in toSterling Selling and Fulfillment Foundation and establish their own personalized session. When the user signs in, the following properties are set up for a session:

- Locale settings
- Security access controls
- User properties such as preferred theme, organization, and so forth

Customizing the login.jsp

About this task

You can customize the login.jsp page to accept additional inputs from the user in the Sign In screen. By default, the Sign In screen accepts two fields, user name and password. After adding the required additional fields, you can validate or authenticate the newly added fields.

To customize the login.jsp:

Procedure

1. To accept the additional fields during login, edit the logininputs.jsp file and add the required additional fields after the user name and password field definitions.
2. To validate or authenticate the newly added fields, in your custom PostAuthentication class, implement the doPostAuthenticate() method of the IYFSPostAuthentication interface. The doPostAuthenticate() method takes the HttpServletRequest object as a parameter, through which you can access the additional fields from the logininputs.jsp. Errors, if any, should be wrapped in

APIManager.XMLExceptionWrapper exception and thrown from this method. If there are no errors, the doPostAuthenticate() method returns True.

By default, the HTML UI framework authenticates the user name and password. If the user name or password fails, a "Login Failed" message is displayed. In the same way, if the additional field authentication fails, a "Login Failed" message is displayed. However, if the post authentication class is unavailable, "Application error, please contact administrator" is displayed.

3. In the application's config.xml file, add the <Context-Params> tag and under the <Context-Params> tag add the <Context-Param> tag for defining the parameter names of the PostAuthentication class. For example,

```
<WebComponents>
     <ContextParams>
          <Context-Param>
            <Param-Name> PostAuthenticationClass</Param-Name>
            <Param-Value>com.yantra.platformdemo.ui.backend.ClientPostAuthClass
            </Param-Value>
          </Context-Param>
      </ContextParams>
    </WebComponents>
```

Setting Up the Locale

The locale you set up determines the language in which on-screen literals display. The start.jsp file displays the Sign In screen in the default language. After a user logs in, the language displayed on-screen is determined from the settings specified within your locale.

A multinational organization may want to localize several languages. For example, the standard installation is in German and a user's locale is French. In such situations, 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 used.

The user's entire session is displayed in the language specified in the profile. The language displayed changes if the user selects a different locale or the session terminates.

If the user selects a different locale,Sterling Selling and Fulfillment Foundation dynamically changes all 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.

Configuring Logins for a Specific Locale

About this task

If your user community is multilingual,IBM recommends displaying the login page in all languages. This enables the user to select the appropriate language from the Sign In screen.

To display an internationalized Sign In screen:

Procedure

1. Create a corporate HTML page that contains hyperlinks corresponding to each language that you want to display.
2. For each hyperlink, insert the <a href> tag to link to the locale-specific page using the following syntax:

   ```
   /smcfs/console/start.jsp?LocaleCode=locale_code
   ```

   The *locale_code* value must match the entry specified in the Locale Code field in the Applications Manager. For more information about the Locale Code field and its suggested syntax, see the Applications Manager, see the Sterling Selling and Fulfillment Foundation:*Configuration Guide* .

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

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

   This displays the Sign In screen in the specified locale. After you log in, the application opens in the user's default locale.

Configuring the User Sign In from an External Application

When integratingSterling Selling and Fulfillment Foundation with external applications, it may be necessary to enable a user to automatically log in to Sterling Selling and Fulfillment Foundation from an external application. To integrate the two applications, you must configure the link from your external application's portal that automatically connects to Sterling Selling and Fulfillment Foundation.

When integrating two applications, consider the following scenarios:

- Logging in a user and opening the user's home page
- Logging in a user and opening a specific view

To integrate Sterling Selling and Fulfillment Foundation with an external application:

Based on your requirement, you can insert any one of the following URLs within the portal of an external application:

- If you want to launch Sterling Selling and Fulfillment Foundation and open the user's home page, log in as an application user and open the user's home page by using the following syntax:

  ```
  /smcfs/console/start.jsp?UserId=LoginID
  &Password=PassPhrase
  ```

- If you want to launch Sterling Selling and Fulfillment Foundation and open a specific view, log in as an application user and open the view by using the following syntax (illustrates opening the default order search view):

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

In either case, if login fails, the browser opens theSterling Selling and Fulfillment Foundation Sign In screen, which prompts the user to enter Login ID and Password.

Supporting External Authentication

If users must log into Sterling Selling and Fulfillment Foundation transparently using a domain password, the application supports third-party single sign-on applications. In the event your environment does not support a single sign-on application, the application supports external authentication and internal authentication (by default).

For information on using external authentication, see the *Sterling Selling and Fulfillment Foundation: Installation Guide*. For programming information on single sign-on, see the Javadocs.

Chapter 6. Customizing Corporate Logos in the JSP Console

About Corporate Logos

The standard application conveys Sterling Selling and Fulfillment Foundation 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

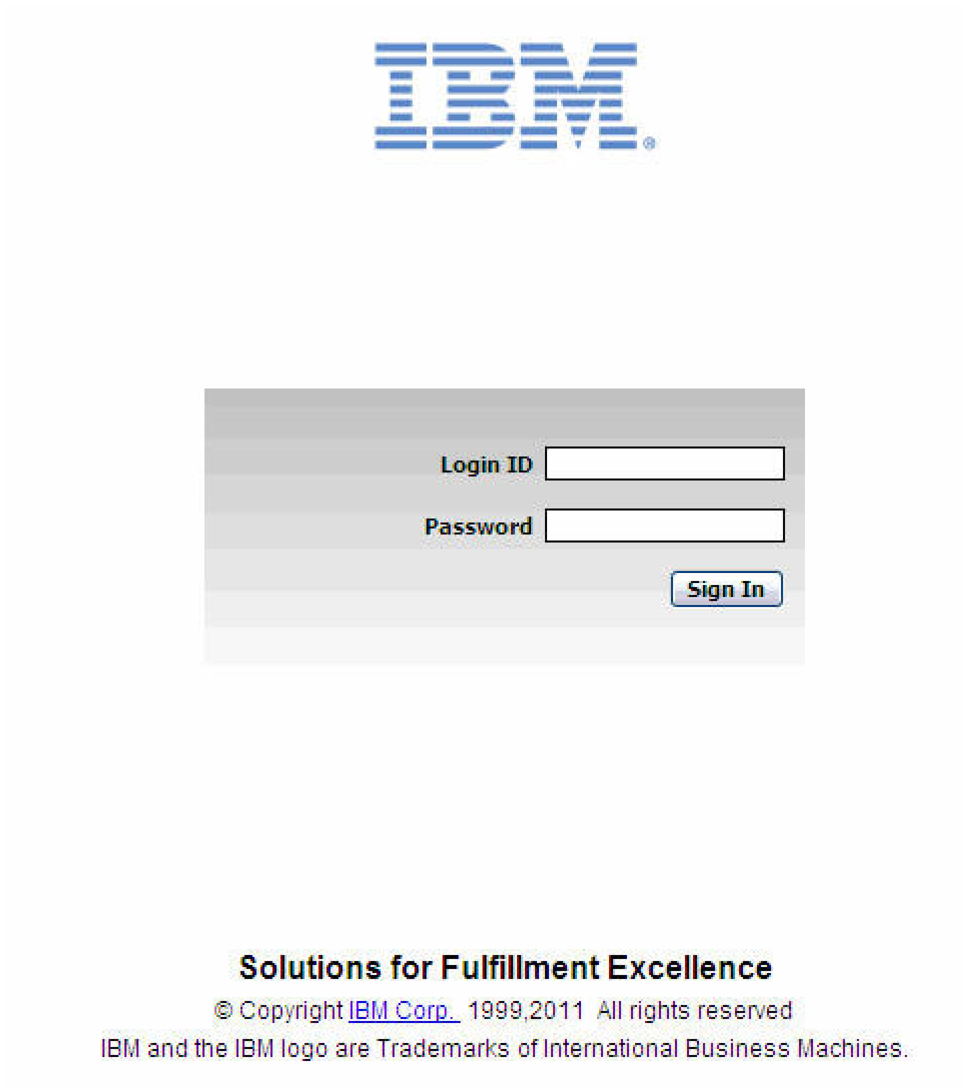
Use a unique image in each instance, since they all use images of different sizes.

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 Hierarchy tree for a console resource is /smcfs/console/icons/ consoleresource.jpg.

Customizing the Logo on the Sign In Screen

About this task

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



Solutions for Fulfillment Excellence

© Copyright IBM Corp. 1999,2011 All rights reserved

IBM and the IBM logo are Trademarks of International Business Machines.

To customize the logo on the Sign In screen:

Procedure

1. Copy the *INSTALL_DIR*/repository/eardata/platform/war/console/logindetails.jsp file to *INSTALL_DIR*/extensions/global/webpages/console/logindetails.jsp file.

   If the /global/webpages/console/ directory structure does not exist, create the required directory structure.
2. Open the new logindetails.jsp file and do the following:
   a. Search for YANTRA_LOGIN_RIGHT to find the <img> tag referring to theIBM logo. Change the <img> tag to point to your corporate logo.

      For example, for MyLogo.jpg you would use:
      ```
      <TR>
                <TD align=center valign=bottom>
                  <img src="<%=request.getContextPath()%>/extn/console/icons/MyLogo.jpg"/>
                </TD>
               </TR>
      ```
   b. Search for YANTRA_ICON to find the <link> tag referring to theIBM icon. Change the <link> tag to point to your corporate icon.

For example, for MyIcon.jpg you would use:

```
<head>
          <link REL="SHORTCUT ICON"
            HREF="<%=request.getContextPath()%>/extn/console/icons/MyIcon.jpg"/>
          <title><yfc:i18n>Yantra_7x</yfc:i18n></title>
         </head>
```

3. Copy your image files to the *INSTALL_DIR*/extensions/global/webpages/console/icons/ directory.

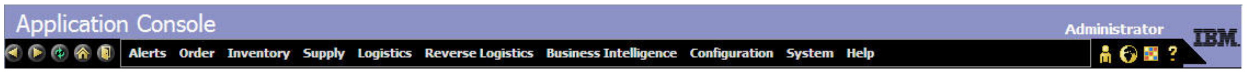
   If the /global/webpages/console/icons/ directory structure does not exist, create the required directory structure.
4. Rename *INSTALL_DIR*/repository/eardata/smcfs*application_name*/extn/web.xml.sample file to *INSTALL_DIR*/repository/eardata/smcfs*application_name*/extn/web.xml. Add the entry for both the image URIs that you specified in Step 2. For example, the web.xml entries are as follows:

```
<context-param>
   <param-name>bypass.uri.extn1</param-name>
   <param-value>/console/icons/MyLogo.jpg</param-value>
</context-param>
<context-param>
   <param-name>bypass.uri.extn2</param-name>
   <param-value>/console/icons/MyIcon.jpg</param-value>
</context-param>
```

Customizing the Logo on the Menu Bar

About this task

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



When a user logs in to Sterling Selling and Fulfillment Foundation, 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.

To customize the logo on the Menu Bar:

If you have not enabled the skins, perform the following tasks:

Procedure

1. Copy the *INSTALL_DIR*/repository/eardata/platform/war/common/menubar.jsp file to *INSTALL_DIR*/extensions/global/webpages/common/menubar.jsp.

   If the /global/webpages/common/ directory structure does not exist, create the required directory structure.
2. Open the new menubar.jsp file and search for YANTRA_LOGO to find the <img> tag referring to theIBM logo. Change the <img> tag to point to your corporate logo, specifying the path as ="../console/icons/ (for example, for MyLogo.jpg you would use ../console/icons/MyLogo.jpg).

3. Copy your image file to the *INSTALL_DIR*/extensions/global/webpages/icons/console/icons/ directory.

   If the /global/webpages/icons/console/icons/ directory structure does not exist, create the required directory structure.
4. From the *INSTALL_DIR*/extensions/global/webpages/icons/ directory, archive the entire console directory into a yantraiconsbe.jar file.

Customizing the Logo on the Menu Bar with Skins

About this task

If you have enabled the skins, perform the following tasks:

Procedure

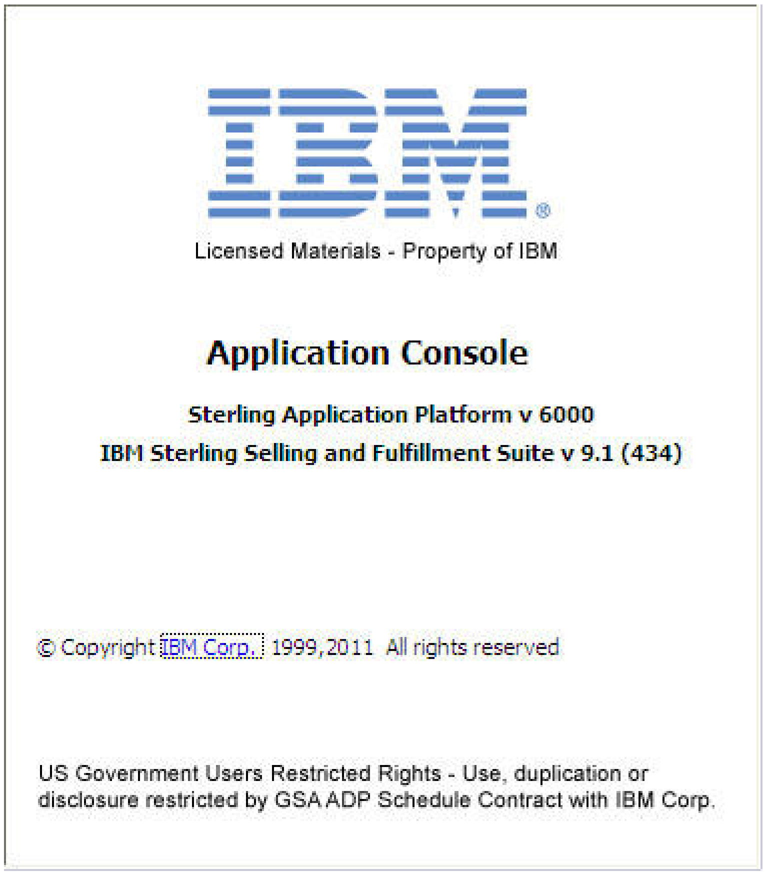
1. Copy the *INSTALL_DIR*/repository/eardata/platform/war/skins/*skin-name*/common/menubar_anchor.jsp file to *INSTALL_DIR*/extensions/global/webpages/skins/*skin-name*/common/menubar_anchor.jsp.

   If the /global/webpages/common/ directory structure does not exist, create the required directory structure.
2. Open the new menubar_anchor.jsp file and do the following:
   a. Search for logo.jpg to find the <img> tag referring to theIBM logo. Change the <img> tag to point to your corporate logo, specifying the path as `="../console/icons/` (for example, for MyLogo.jpg you would use `../console/icons/MyLogo.jpg`).
   b. Modify the CSS class defined for the logo.jpg <img> tag as per your requirement.
3. Copy your image file to the *INSTALL_DIR*/extensions/global/webpages/icons/console/icons/ directory.

   If the /global/webpages/icons/console/icons/ directory structure does not exist, create the required directory structure.
4. From the *INSTALL_DIR*/extensions/global/webpages/icons/ directory, archive the entire console directory into a yantraiconsbe.jar file.

Customizing the Logo in the About Box

The About Box indicates the version number of an application.Sterling Selling and Fulfillment Foundation ships a standard About Box that a user can access by clicking the logo displayed on the Menu Bar shown in. The following figure shows the Application Console About Box.


Licensed Materials - Property of IBM

Application Console

Sterling Application Platform v 6000

IBM Sterling Selling and Fulfillment Suite v 9.1 (434)

© Copyright IBM Corp. 1999,2011 All rights reserved

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

When displaying the About Box, the application reads the version number based on predetermined 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 IBM, 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.

Customize the Logo on the About Box with Skins Not Enabled

About this task

To customize the logo on the About Box:

If you have not enabled the skins, perform the following tasks:

Procedure

1. Copy the *INSTALL_DIR*/repository/eardata/platform/war/console/about.jsp file to *INSTALL_DIR*/extensions/global/webpages/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.

Customizing the Logo in the About Box with Skins

About this task

If you have enabled the skins, perform the following tasks:

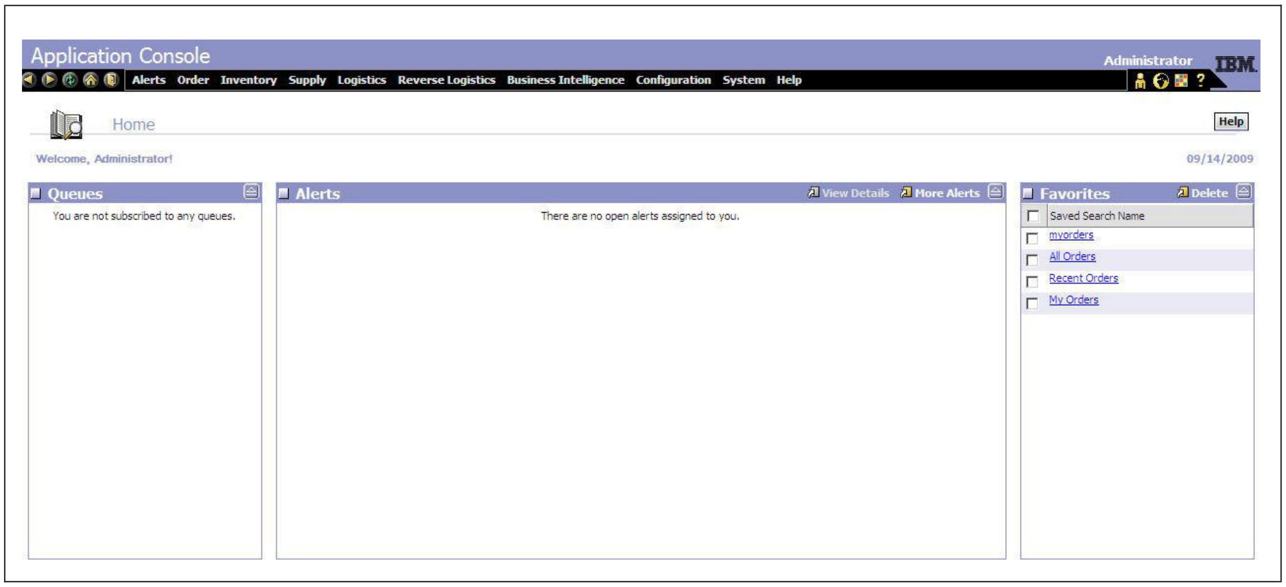
Procedure

1. Copy the *INSTALL_DIR*/repository/eardata/platform/war/console/about_mb.jsp file to *INSTALL_DIR*/extensions/global/webpages/console/about_mb.jsp.
2. Write your own HTML in your new about_mb.jsp page.
3. In order to change the size and positioning of the About Box pop-up window, modify your copy of the menubar_anchor.jsp file and look for the popupAboutBox_mb() function. Modify the properties such as height, width, left, and top as per your needs.

Chapter 7. Screens in the JSP Console

About JSP Console Screens

In the Application Console, a screen is made up of a container page which defines how the screen should display any inner panels it contains. Within the Sterling Selling and Fulfillment Foundation, anchor pages, inner panels, and the logic associated with them are referred to as views. The following figure depicts the standard Home Page, which shows the components used in a search view.


Application Console
Administrator
IBM
Alerts Order Inventory Supply Logistics Reverse Logistics Business Intelligence Configuration System Help
Home
Help
Welcome, Administrator!
09/14/2009
Queues
You are not subscribed to any queues.
Alerts
View Details
More Alerts
There are no open alerts assigned to you.
Favorites
Delete
Saved Search Name
myorders
All Orders
Recent Orders
My Orders

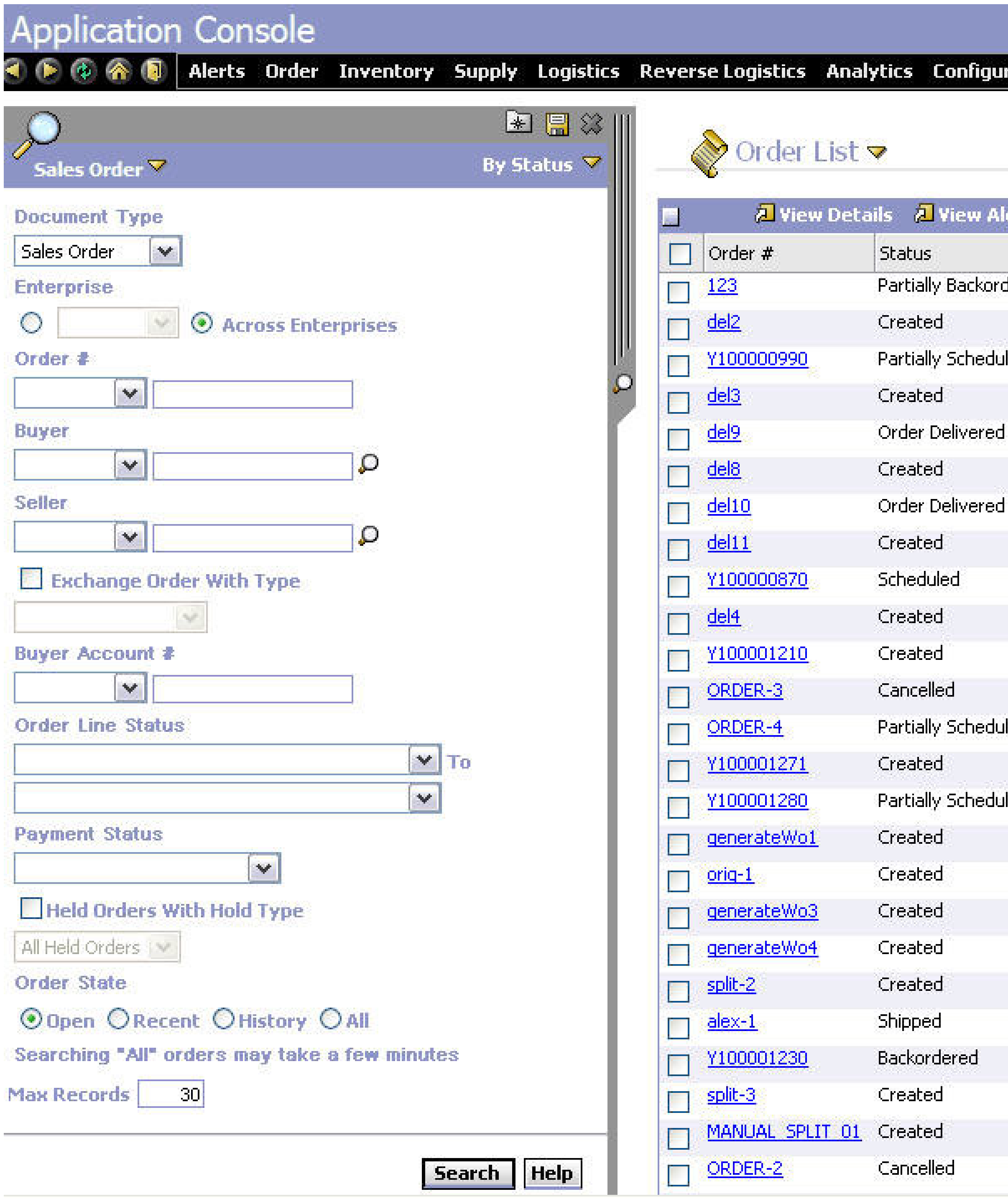
The following types of views are available in the Application Console:

- Search view
- List view
- Detail view

For detailed descriptions of these views, see "Which Screens Can Be Extended?"

Screen Layout in the JSP Console

All screens follow the same overall organizational pattern. In general, application screens are laid out with a Menu Bar at the top and side-by-side search and list views below it. The following figure shows the typical organization of a screen.



Document Type: Sales Order

Enterprise: [✓ Across Enterprises]

Order #

Buyer

Seller

Exchange Order With Type

Buyer Account #

Order Line Status To

Payment Status

Held Orders With Hold Type: All Held Orders

Order State: [✓ Open] Recent History All

Searching "All" orders may take a few minutes

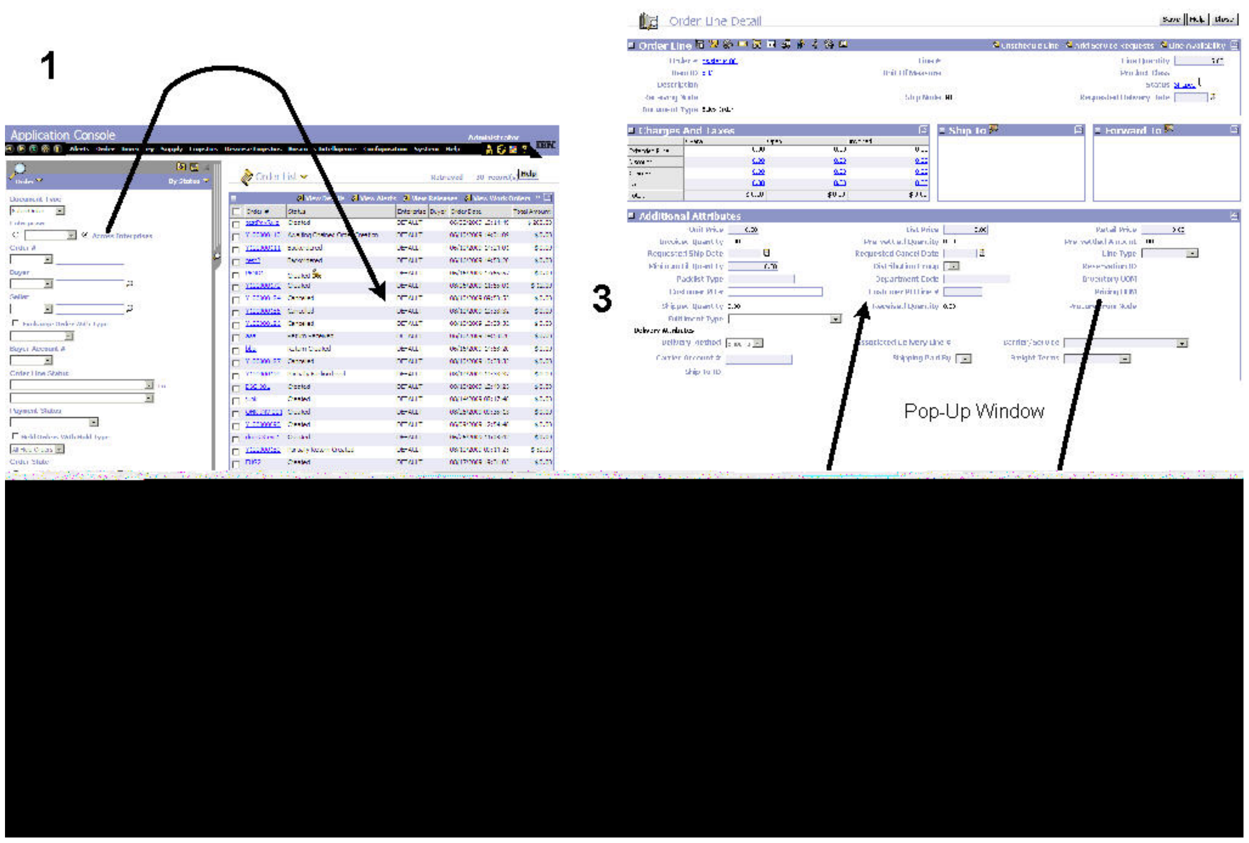
Max Records 30

Order #	Status
123	Partially Backord
del2	Created
Y100000990	Partially Schedul
del3	Created
del9	Order Delivered
del8	Created
del10	Order Delivered
del11	Created
Y100000870	Scheduled
del4	Created
Y100001210	Created
ORDER-3	Cancelled
ORDER-4	Partially Schedul
Y100001271	Created
Y100001280	Partially Schedul
generateWo1	Created
orig-1	Created
generateWo3	Created
generateWo4	Created
split-2	Created
alex-1	Shipped
Y100001230	Backordered
split-3	Created
MANUAL_SPLIT_01	Created
ORDER-2	Cancelled

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.

Screen Navigation in the JSP Console

TheSterling Selling and Fulfillment Foundation screen navigation behavior follows a standard, consistent pattern. The following figure shows the flow of navigation logic from one inner panel to the next.



- The search view fills the left side of the screen.
- 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.
- 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.
- Closing the pop-up window returns the focus to previous view.

Customizing Screen Navigation

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

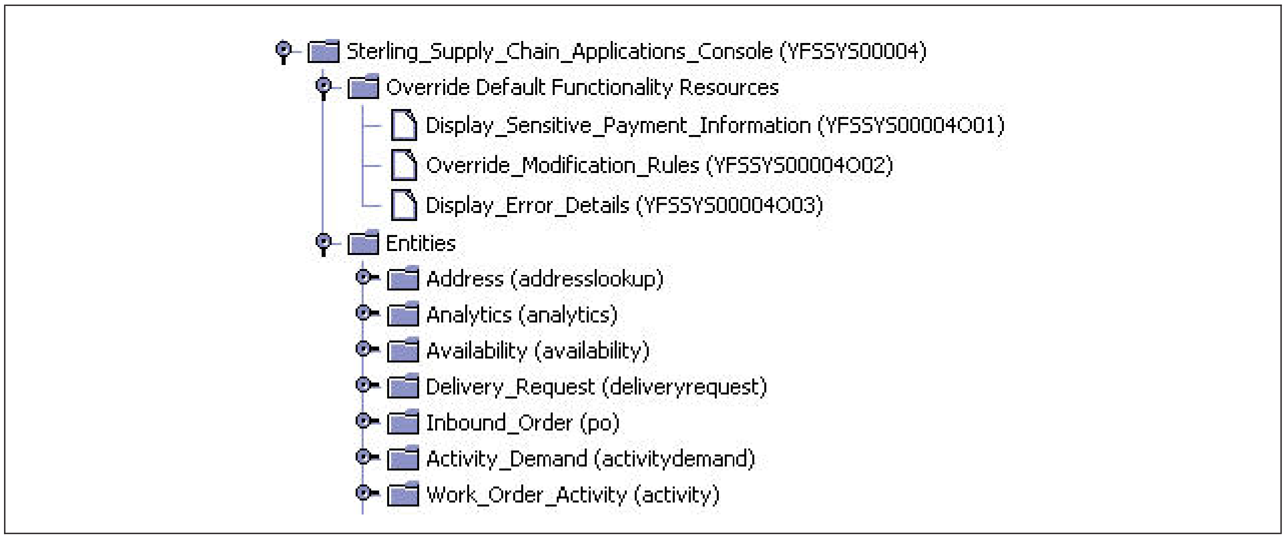
Which Screens Can Be Extended?

The screens contained within an inner panel all have business and programmatic logic associated with them. 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 Selling and Fulfillment Foundation contains 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 Hierarchy tree enables you to customize and create entities that suit your business needs. The following figure shows how entities appear in the Resource Hierarchy tree. The hierarchical order is based on the default order of navigation.


Sterling_Supply_Chain_Applications_Console (YFSSYS00004)
Override Default Functionality Resources
Display_Sensitive_Payment_Information (YFSSYS00004O01)
Override_Modification_Rules (YFSSYS00004O02)
Display_Error_Details (YFSSYS00004O03)
Entities
Address (addresslookup)
Analytics (analytics)
Availability (availability)
Delivery_Request (deliveryrequest)
Inbound_Order (po)
Activity_Demand (activitydemand)
Work_Order_Activity (activity)

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 Hierarchy tree.

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.

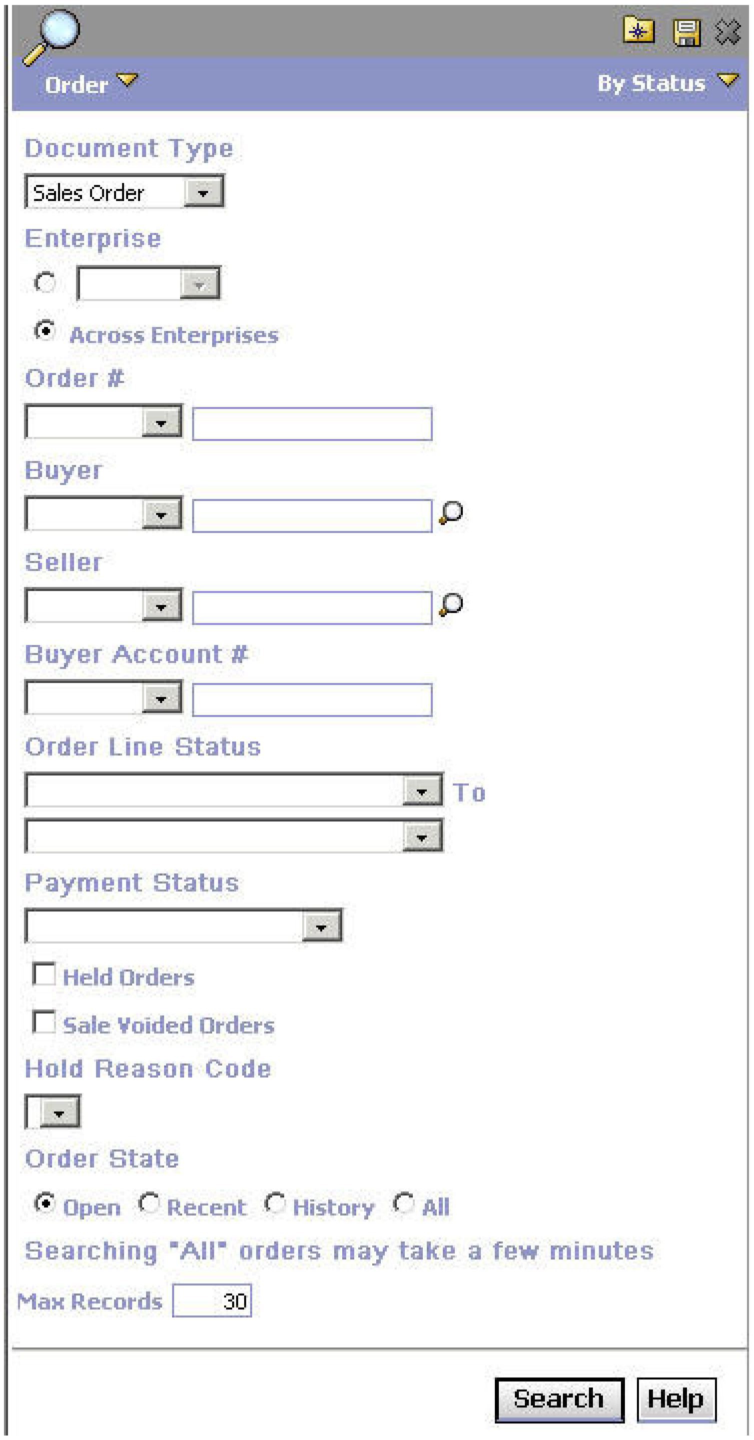
You can prevent this default API from being called for a specific view by selecting the Ignore Default API parameter in the resource configuration screen.

For more information about resource sequencing, see the Sterling Selling and Fulfillment Foundation: *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.

Search View Screens in the JSP Console

A search view contains a list of criteria from which to query. The following figure shows a standard search view.


Order ▼ By Status ▼

Document Type

Sales Order

Enterprise

○

[✓ Across Enterprises]

Order #

Buyer

Seller

Buyer Account #

Order Line Status

To

Payment Status

☐ Held Orders

☐ Sale Voided Orders

Hold Reason Code

Order State

[✓ Open] ○ Recent ○ History ○ All

Searching "All" orders may take a few minutes

Max Records 30

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

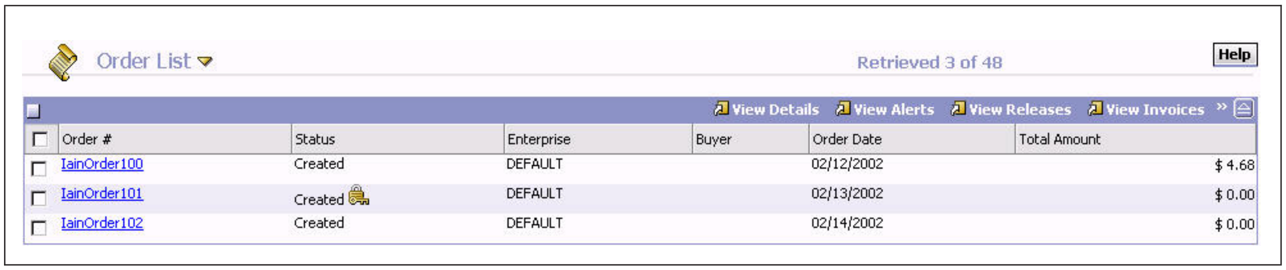
- JSP pages - render the screen. For detailed information on JSPs, see "JSP Functions for the Console JSP Interface."
- APIs - populate drop-down menus. For each API, you can specify input parameters and an output template. For more information on configuring API resources used by the Application Console, see the Sterling Selling and Fulfillment Foundation.

List View Screens in the JSP Console

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

The following figure shows a standard list view.



Order #	Status	Enterprise	Buyer	Order Date	Total Amount
IainOrder100	Created	DEFAULT		02/12/2002	$ 4.68
IainOrder101	Created	DEFAULT		02/13/2002	$ 0.00
IainOrder102	Created	DEFAULT		02/14/2002	$ 0.00

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 *INSTALL_DIR*/properties/customer_overrides.properties file as described in "Maximum Records For List Views in the Console JSP."

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

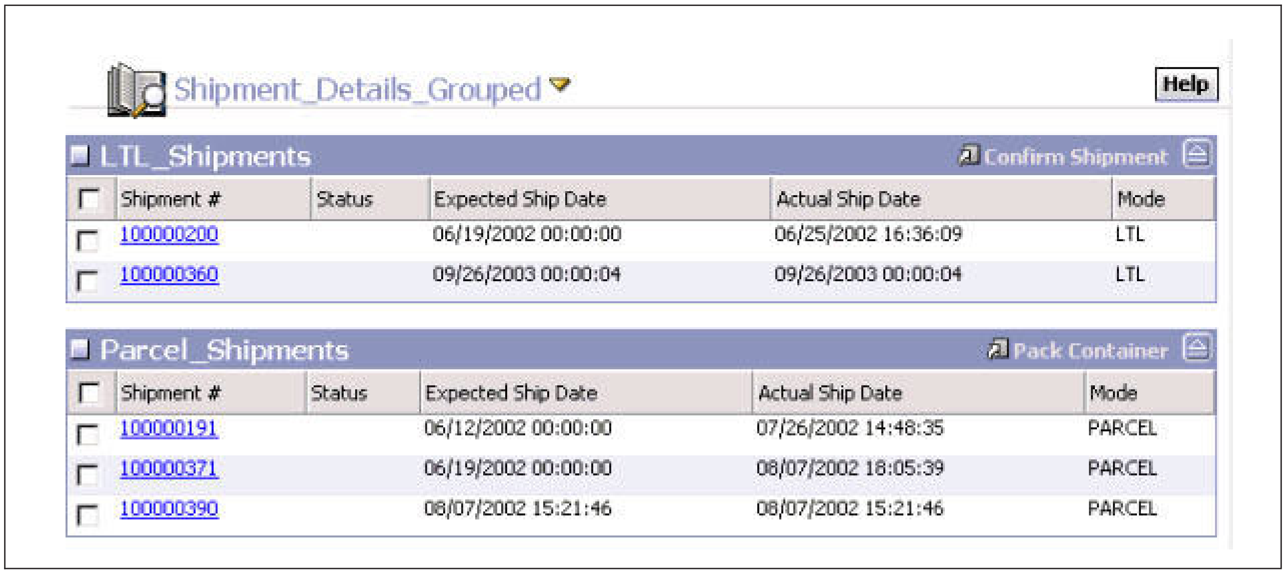
- JSP pages - render the body. It is the only part of the view that does not belong to the Presentation Framework.
- Additional APIs - are called for that list view.
- Actions - See “Actions from List and Detail Views in the JSP Console."

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.

The following figure 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.


Shipment_Details_Grouped

LTL_Shipments

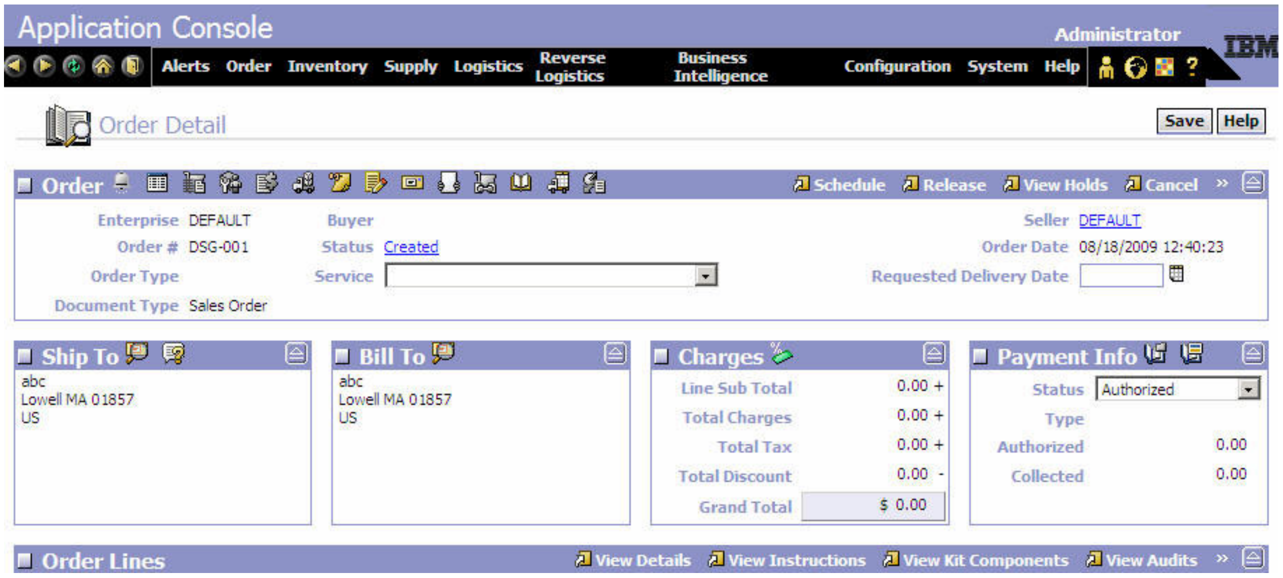
	Shipment #	Status	Expected Ship Date	Actual Ship Date	Mode
	100000200		06/19/2002 00:00:00	06/25/2002 16:36:09	LTL
	100000360		09/26/2003 00:00:04	09/26/2003 00:00:04	LTL

Parcel_Shipments

	Shipment #	Status	Expected Ship Date	Actual Ship Date	Mode
	100000191		06/12/2002 00:00:00	07/26/2002 14:48:35	PARCEL
	100000371		06/19/2002 00:00:00	08/07/2002 18:05:39	PARCEL
	100000390		08/07/2002 15:21:46	08/07/2002 15:21:46	PARCEL

Detail View Screens in the JSP Console

A detail view shows more specific information about an entity. The following figure shows a standard detail view.


Order Detail

Enterprise	DEFAULT	Buyer		Seller	DEFAULT
Order #	DSG-001	Status	Created	Order Date	08/18/2009 12:40:23
Order Type		Service		Requested Delivery Date	
Document Type	Sales Order				

Ship To: abc, Lowell MA 01857, US

Bill To: abc, Lowell MA 01857, US

Charges

Line Sub Total	0.00 +
Total Charges	0.00 +
Total Tax	0.00 +
Total Discount	0.00 -
Grand Total	$ 0.00

Payment Info

Status	Authorized
Type	
Authorized	0.00
Collected	0.00

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 vertically one after another.

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 Selling and Fulfillment Foundation: 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 Hierarchy tree.
- Save Action—one or more. See "Actions from List and Detail Views in the JSP Console."

Wizards in the JSP Console

A wizard view is used to accept multiscreen inputs. A wizard view consists of one or more wizard pages. Each wizard page is defined as a detail view. For more information about detail views, see "Detail View Screens in the JSP Console."

Wizard view is permission controlled, priority driven, and contains wizard definition XML. A wizard view consists of the following components:

- Wizard Definition—A wizard definition contains one or more wizard entity (wizard page or wizard rule) definitions and one or more wizard transition definitions. A wizard definition defines the flow of the wizard. For more information about wizard definition, see "What is a Wizard Definition?"
- Wizard Servlet—A wizard servlet handles the following wizard-related tasks:
  - Determining the wizard view permission
  - Calculating the wizard rule from the wizard definition XML
  - Handling the data posted by the previous wizard page
  - Calling Detail servlet to determine the detail view for the current wizard page and calling the APIs corresponding to that detail view
- Wizard Anchor Page—A wizard anchor page provides placeholders for including breadcrumbs and wizard action buttons. The wizard anchor page also includes the JSP of the detail view corresponding to the current wizard page.
- Save Action—The Save element contains the list of all the APIs to be called on save action, the JavaScript handler on Finish button, and the view ID of the view that will be shown when the wizard is finished.
- Wizard Actions—You can fire four actions from a wizard view. For more information about type of actions for a wizard view, see "Actions in Wizards."

What are Breadcrumbs?

The breadcrumbs show users their location in the wizard view. They show the path traversed by the user in the wizard view up to the current page. Breadcrumbs also allow users to go back to each of the previous pages the user has navigated through in order to get to the current page. You can define the breadcrumbs category for each wizard page in the wizard definition. Breadcrumb categories are used to determine the path traversed, and are shown in the breadcrumbs.

What is a Wizard Definition?

A wizard definition defines the flow of the wizard. A wizard definition consists of one or more wizard entities (wizard page or rule) and wizard transitions. 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 wizard transition value. Wizard transition is used to transfer control from one wizard entity to another. A Wizard definition contains:

- Wizard Entity—There are two types of wizard entities:
  - Wizard Page—A wizard page takes care of the UI in order to take the inputs from a user. Each wizard page is defined as a detail view and has a view ID associated with it. A wizard definition can have one or more wizard pages. Each wizard page can also specify a JavaScript that will be called on the Next action. You can also specify the breadcrumb for each wizard page. A wizard definition can have one or more wizard pages.
  - Wizard Rule—A wizard rule is a logical step that performs some computations to evaluate different output values. Based on these output values, the flow of the wizard is decided. You can define both Java and Greex rules. A wizard definition can have one or more wizard rules.
- Wizard Transition—A wizard transition is used to transfer control from one wizard entity (wizard page or rule) to another wizard entity (wizard page or rule). Wizard transition connects sequences of wizard entities with each other. The wizard transition value is compared with the output of the wizard rule and, based on this comparison, the control is transferred to the next wizard entity (wizard page or rule). A wizard definition can have one or more wizard transitions.

Actions in Wizards

From the wizard view, the following actions are possible:

- Next—On the Next action, the JavaScript performs all the validations and posts the data with an additional parameter mentioning the type of action. The Wizard servlet executes the rules and determines the next wizard page for the current wizard view. The Detail servlet is called to display the next wizard page for the current wizard view. Data from all the earlier wizard pages is merged and made available on request.
- Previous—On the Previous action, the Wizard servlet determines the previous wizard page for the current wizard view and removes all the parameters posted by that wizard page from the session. The Detail servlet is called to display the previous wizard page for the current wizard view.
- Finish—On the Finish action, the Wizard servlet calls the Save action defined for the current wizard view.
- Cancel—On the Cancel action, the Wizard servlet purges the complete wizard data from the session.

All the wizard actions post the data back to the server and the wizard servlet processes, and determines the wizard flow. Wizard actions send an additional parameter with a request mentioning the action type.

You can define JavaScript on save resource for the Finish action and the Next action on each wizard page. These JavaScripts can perform extra validations.

Actions from List and Detail Views in the JSP Console

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 a line without committing the transaction in the database. This option can be enabled in the Resource Hierarchy tree. For more information on enabling this option refer to the Applications Manager, see the Sterling Selling and Fulfillment Foundation:*Configuration Guide* .

You can use each action by itself but they are more useful when combined. 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.

Configuring a View to Open Only on the Success of an API

About this task

If you wantSterling Selling and Fulfillment Foundation to open a view only when an API returns success, you must configure that specifically.

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

Procedure

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 application's 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.

Chapter 8. Customizing Views and Wizards in the JSP Console

Creating a View without a Template in the JSP Console

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

Creating a New Resource

About this task

To create a new resource:

Procedure

1. From the Applications Manager menu select **Application Platform** → **Presentation** → **Resources**. This displays the Resource Hierarchy tree.
2. Select **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, 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

What to do next

Alternatively, if you want an easier way to customize Sterling Selling and Fulfillment Foundation, you can create new views, actions, or icons within the entities available in the Application Console, using templates.

Creating a View with a Template in the JSP Console

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, use the *INSTALL_DIR*/properties/customer_overrides.properties file to set the yfs.uidev.refreshResources property to Y. Ensure that you set the property before proceeding to click the refresh icon.

Note: For additional information about overriding properties using the customer_overrides.properties file, see the Sterling Selling and Fulfillment Foundation: Properties Guide.

Note: When creating new views, consider ease of maintenance as well as ease of creation. For more information, see "Before You Customize the JSP Console."

Customizing a Search View in the JSP Console

About this task

Customizing a search view involves minimal changes.

To customize a search view:

Procedure

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 Hierarchy tree, 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 *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ directory.

   Note: Customizing this view may require other files to be copied into the /webpages 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 Hierarchy tree, navigate to your new view. Enter the relative path to your JSP file in the JSP field. For example:
   ```
   /extensions/global/webpages/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.
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 select the Refresh Entity Cache icon.
   - If you want to update all resources - Select the Refresh Cache icon.
9. Log in toSterling Selling and Fulfillment Foundation again to test your changes.

Customizing a Regular List View in the JSP Console

About this task

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:

Procedure

1. Navigate to the screen that you want to change to 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 Hierarchy tree, 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 *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ directory.

   Note: Customizing this view may require other files to be copied into the webpages 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 Hierarchy tree, navigate to your new view. Enter the relative path to your JSP file in the JSP field. For example:
   ```
   /extensions/global/webpages/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.
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 select the Refresh Entity Cache icon.
   - If you want to update all resources, select the Refresh Cache icon.
9. Log intoSterling Selling and Fulfillment Foundation again to test your changes.

Customizing an Advanced List View in the JSP Console

About this task

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:

Procedure

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

   When the user runs 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.

   When using the following controls, make sure to include the corresponding JSP functions of your application when binding the input fields on the search view to an XML. This ensures 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, verify that the controls use the functions listed here.

Control	Function
Textbox	getTextOptions
Select dropdown	getComboOptions
Radio button	getRadioOptions
Check Box	getCheckBoxOptions
Hidden Inputs	getTextOptions

3. Create a custom detail JSP page as described in "Customizing Detail Views in the JSP Console." (This is used as the advanced list view.)
4. 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.
5. Determine how the advanced list view should open.

Maximum Records for List Views in the Console JSP

By default, the application displays 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 customer_overrides.properties file. Once you set a new upper limit in the customer_overrides.properties file, this system-level setting applies to all users.

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

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

Modifying the Maximum Number of Records

About this task

To modify the maximum number of records displayed:

Configure the yfs.ui.MaxRecords property in the *INSTALL_DIR*/properties/customer_overrides.properties file.

Customizing Detail Views in the JSP Console

About this task

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:

Procedure

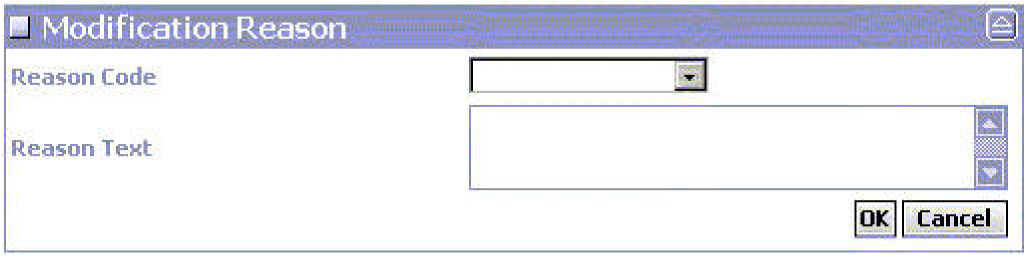
1. Navigate to the screen that you want to change, so you can determine its view ID.
2. Hover your mouse pointer over the screen's view title. The view title's tool tip indicates the view ID.
3. From the Resource Hierarchy tree, navigate to the view ID and copy it, including its sub-resources, when prompted.
4. Copy the JSP file for the original view into the *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ 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 Hierarchy tree 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.

   Note: Verify that the anchor page contains the Resource IDs of all of the inner panels you want included. See the *Sterling Selling and Fulfillment Foundation: 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 *INSTALL_DIR*/extensions/global/webpages/*PathOfOriginalJSP*/ directory.
   b. From the Resource Hierarchy tree, navigate to your new inner panel. Enter the relative path to your JSP file in the JSP field. For example:
      ```
      /extensions/global/webpages/om/order/detail/order_detail_header.jsp
      ```
   c. Edit your inner panel JSP file as needed.
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 Customized Views Across the Application."
10. 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 select the Refresh Entity Cache icon.
    - If you want to update all resources, select the Refresh Cache icon.
11. Log into the application again to test your changes.

Blocking Reason Code Pop-ups in Detail Views

The SAVE action for the changes you make in the Detail Screen ofSterling Selling and Fulfillment Foundation console results in a pop-up asking for the Reason Code for the changes made. The Reason Code pop-up can be blocked from appearing only for a custom screen.



Note: Pop-up editing is possible only if the screen is a custom screen. Default screens are not editable.

Removing the Appearance of a Pop-up Screen

About this task

To remove the appearance of a pop-up screen invoked by the 'SAVE' action:

Procedure

1. From the Applications Manager menu, select **Application Platform** → **Presentation** → **Resources**. This displays the Resource Hierarchy tree.
2. Select **Entity** → **Detail View** → **Action** that you want to edit. In this case, **SAVE** is chosen as the action to be modified.
3. Remove the JavaScript method that invokes the pop-up screen.

About Customizing Wizards in the JSP Console

A wizard view supports multiscreen inputs. It consists of one or more wizard pages. Each wizard page is defined as a detail view to show more specific information. A wizard view can make use of multiple APIs. You can customize a wizard view by either customizing the single wizard page or by customizing the wizard definition.

Customizing Wizard Definitions in the JSP Console

A wizard definition controls the flow of the wizards through wizard rule evaluation. The wizard definition is defined in the application.xml file. It contains information about an individual wizard page and rule implementations. You can customize the wizard definition to include or exclude one or more wizard pages. You can also add or modify rule implementations.

Chapter 9. Customizing JSP Files in the JSP Console

About JSP Files in the JSP Console

JSP files contain the syntax that enables 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 the table.

JSP Files for Detail Views

A JSP file for a detail view is typically the most complicated, since detail views 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

Remember: 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.

A Sample JSP File in the JSP Console

The JSP files provided bySterling Selling and Fulfillment Foundation cover every typical use case scenario that you 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 nits of code together. In addition to the JSP file template, see the *User Interface Style Reference* topic for more technical details.

The following example shows some code from a typical order JSP file, which can be used to render an Order list view.

```
<%@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="/smcfs
/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 Presentation Framework, most JSP files only require a reference to the *INSTALL_DIR*/repository/eardata/platform/war/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 *INSTALL_DIR*/repository/eardata/platform/war/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 for the Console JSP Interface."

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

Common JSP for UI Views Across Document Types

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 include 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 is a primary important field)

Using the common_fields.jsp

The common_fields.jsp (located in the *INSTALL_DIR*/repository/eardata/smcfs/war/yfsjspcommon 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."

The following indicates all of the valid parameters that can be passed to the 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.

Parameter	Description
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. Default: false.
RefreshOnEnterpriseCode	Indicates whether the entire screen should refresh when an enterprise is selected. Default: false.
RefreshOnNode	Indicates whether the entire screen should refresh when a node is selected. 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.
ColumnLayout	Indicates the number of columns 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 refreshes 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.

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 by which you can search. 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 passes 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, 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

Other Common Field Features/Notes

The other features and notes of Common Field JSP (common_fields.jsp) are as following:

- 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 appears 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 appears as a combo box. If there are more than 21 records to display, the field appears 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 appears as a protected input that cannot be modified. Otherwise, the node field displays all of the current logged-in organization's owned nodes.

Sample common_fields.jsp for a Search Screen

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 is included:

```
<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>
```

Creating Inner Panels for a Detail View

About Inner panels

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 Presentation Framework provides a default anchor page to do that. In such a case, you do not have to configure any JSP for the detail view.

The following shows the typical syntax for 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 the application 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 you specify here are automatically available to the JSP configured against the resource ID of the inner panel being included.

Steps To Create an Inner Panel

About this task

To create an inner panel:

Procedure

1. Customize the view to which you want to add the inner panel.
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 is *<view ID's resource ID><Suffix>*. For example, I01.

   You only need to refer to the suffix. The Presentation Framework forms the complete inner panel resource ID and includes the appropriate JSP.
3. From the Applications Manager, create the inner panel resources under the newly created view.
4. From the Applications Manager, 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.

Incorporating Customized Views 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 has a sequence number 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.

Chapter 10. Other Customizations in the JSP Console

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. To see the standard Home Page, see the figure in "About JSP Console Screens."

To customize the Home Page, follow either set of steps for customizing a detail view for the Home entity:

- Customizing Detail Views in the JSP Console
- Creating Inner Panels for a Detail View

Customizing Security Servlet Filter for Authenticated Access to URLs

The servlet filter defined for the context-root filters requests for all the content present in the web root, except the login page or pages mentioned in the web.xml file as "bypass.uri". When the servlet filter receives the request for a resource, it validates the request session ID. If the request session ID is valid, it redirects to the particular resource. If the request session ID is invalid, it redirects to the login page. The servlet filter definition is stored in the web.xml file and url-pattern element contains all the URLs that need filter authentication.

If you want to add certain URLs for which the filter authentication should not be applied, add a config-param element for each such URL in the web.xml (located inside your EARFILE/WARFILE/WEB-INF) file as follows:

```
<config-param>
   <param-name>bypass.uri.1</param-name>
   <param-value>/console/login.jsp</param-value>
</config-param>
<config-param>
   <param-name>bypass.uri.2</param-name>
   <param-value>/console/start.jsp</param-value>
</config-param>
<config-param>
   <param-name>bypass.uri.3</param-name>
   <param-value>/console/public/screens</param-value>
</config-param>
```

where param-value element contains the URI for which you do not want filter authentication.

Also, if you want all the files in a particular folder to be bypassed from filter authentication, specify the folder path from the context root as the bypass URI in the param-value element.

Note: Make sure that the value of the param-name element starts with "bypass.uri" string. Also, make sure that the context root is not given in the property file. The context root is dynamically assigned while building an EAR file.

Setting the User Session Timeout

About this task

You can set the session timeout interval (in minutes) for a user by adding an entry for the config-param element in the web.xml file, which is located inside your EARFILE/WARFILE/WEB-INF directory. For example:

```
<session-config>
   <session-timeout>30</session-timeout>
</session-config>
```

Note: The session timeout value defined in the web.xml file will be effective only if the session timeout interval (in seconds) in the YFS_USER table for the corresponding user is equal to or less than 0. By default, in the YFS_USER table, the session timeout value (in seconds) for each user is set to 6000.

Creating a Custom Business Entity

About this task

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

Note: Sterling Selling and Fulfillment Foundation does 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:

Procedure

1. From the Resource Hierarchy tree, 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. Select **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 does not conflict with any resource IDs that might ship with future releases.IBM recommends that you choose any prefix except one that begins with the letter **Y** (which is reserved for use by the application).
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 verify that literals can be localized, use the resource description entered here as the resource bundle key in all of the appropriate resource bundles.
4. Create entries for these newly created resource keys in the *INSTALL_DIR*/extensions/global/resources/extnbundle.properties file and in the corresponding properties files for each locale.

   Note: Ensure that the following file does not exist:

   ```
   INSTALL_DIR/resources/extn/extnbundle.properties
   ```

This file must be removed because it will conflict with the extensions build process for bundle entries.

5. Repeat Step 1 through Step 3 for the Related Entities of the entity being copied.
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 Customized Views Across the Application."

Using Extended Database Columns

You can customize views to incorporate any new columns added to the database.

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

- Search view - "Customizing a Search View in the JSP Console."
- List view - "Customizing a Regular List View in the JSP Console" and "Customizing an Advanced List View in the JSP Console." After following the database extensibility rules, add the field to the output template configured for the API through the Resource Hierarchy tree.
- Detail view - "Customizing Detail Views in the JSP Console."

  After following the database extensibility rules, add the field to the output template configured for the API through the Resource Hierarchy tree.

Using the Override Entity Key Attribute

Often, optimal screen layout dictates the use of an editable list of records. 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. You can override the entity key used for a specific action using the Override Entity Key attribute on an action resource. To construct an input (usually a hidden input or a checkbox) on the JSP, give the input the 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 invoked 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 that lists order lines for an order:

```
<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.

Posting Data for Editable Lists

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

One way 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 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 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 placed 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 Presentation Framework JavaScript function can be used to verify 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.

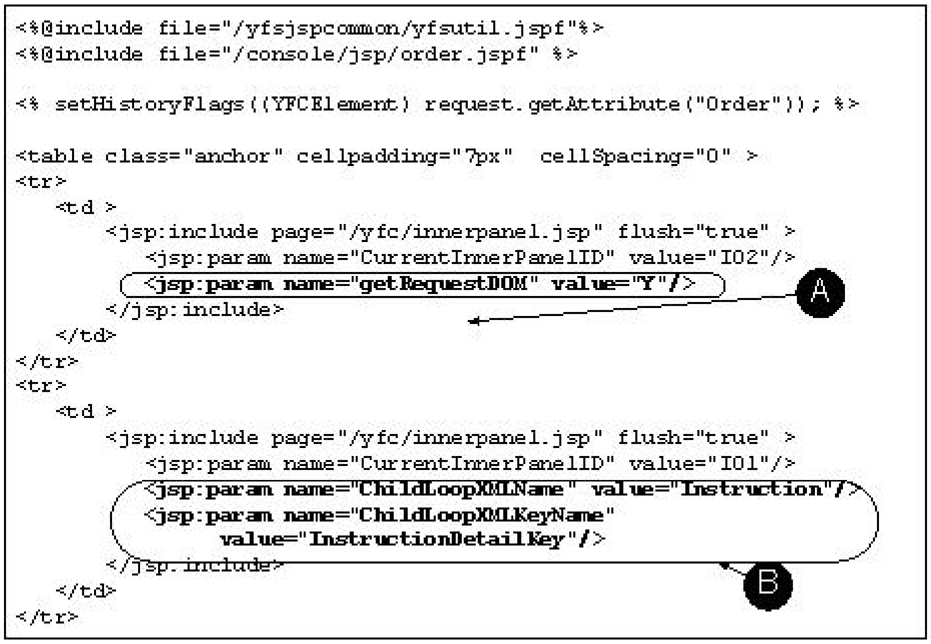
Retaining Unsaved Data in an Editable List

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 refreshes and the data entered in the editable table is 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 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>
```

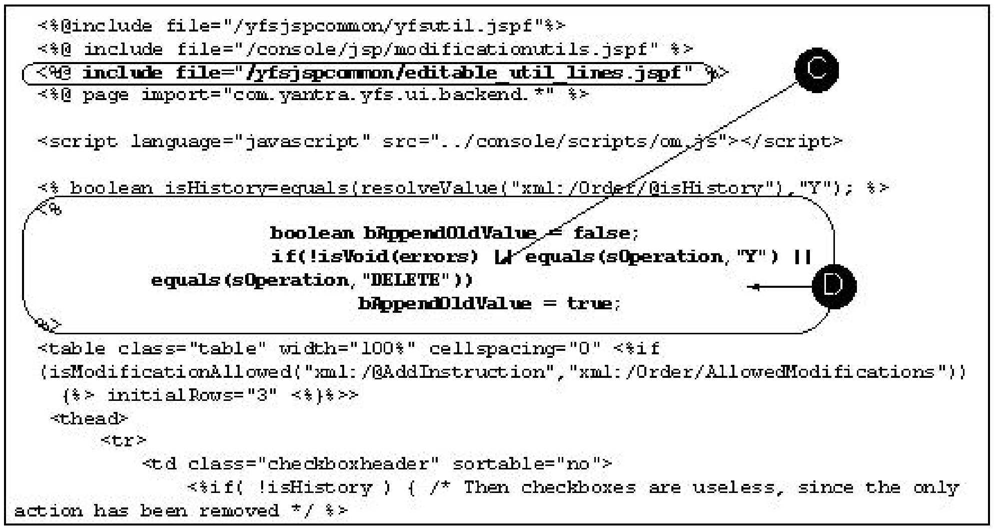
The following table explains the callouts.

Callout	Description
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 are retrieved from the request object and merged with the API output that is used to load the screen initially. The refreshed screen now contains 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 the following table.

These are the parameters to be passed:

Parameter Name	Default Value	Comments
RootNodeName	Order	The root node of the output XML that has the newly entered values 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.

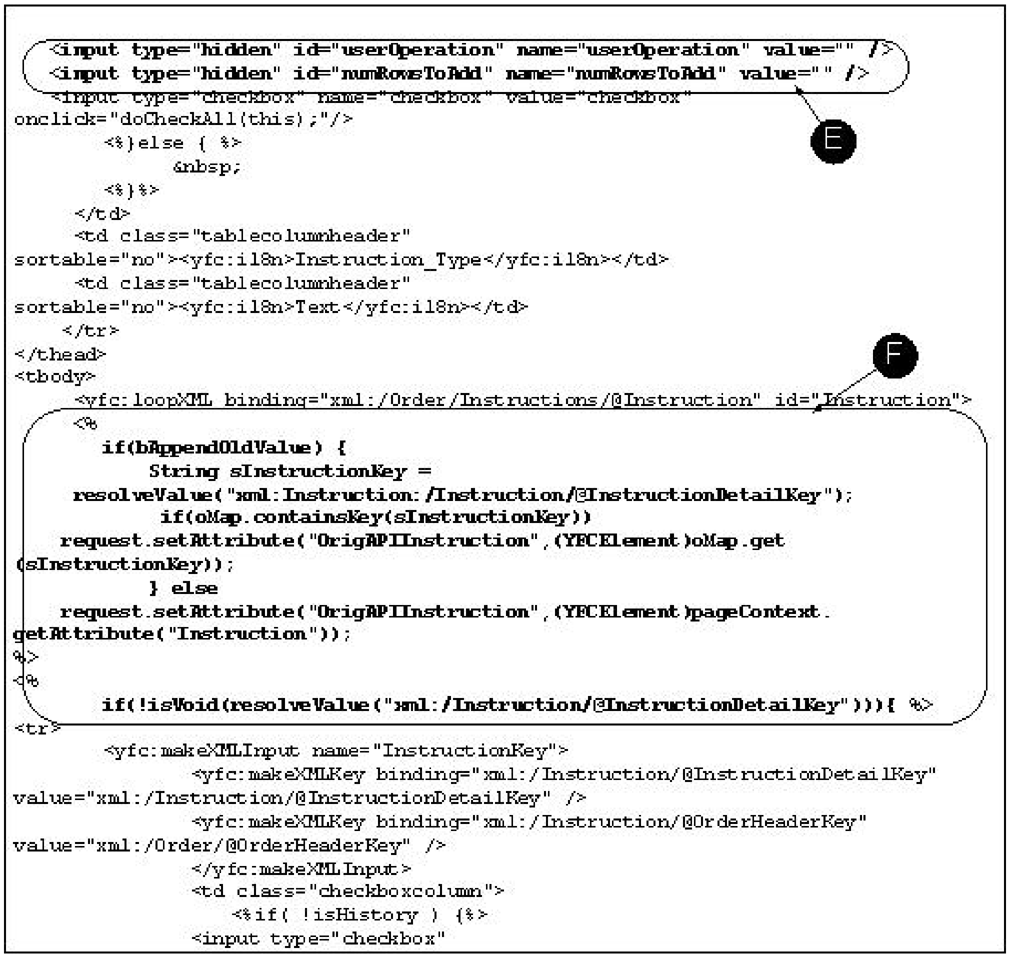
The highlighted code indicates the changes that ensure that the data is retained after an API exception:



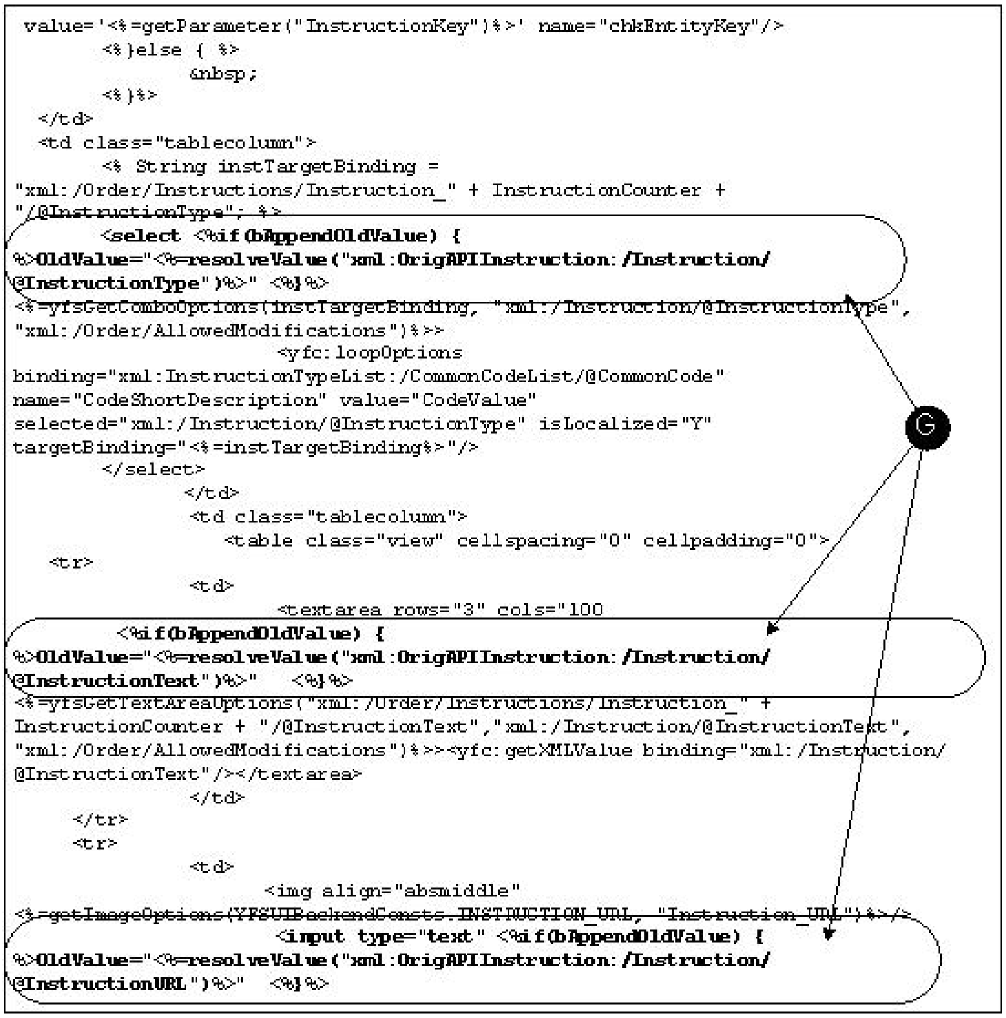
```
  <%@include file="/yfsjspcommon/yfsutil.jspf"%>
  <%@ include file="/console/jsp/modificationutils.jspf" %>
  <%@ include file="/yfsjspcommon/editable_util_lines.jspf" %>
  <%@ page import="com.yantra.yfs.ui.backend.*" %>

  <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 */ %>
```



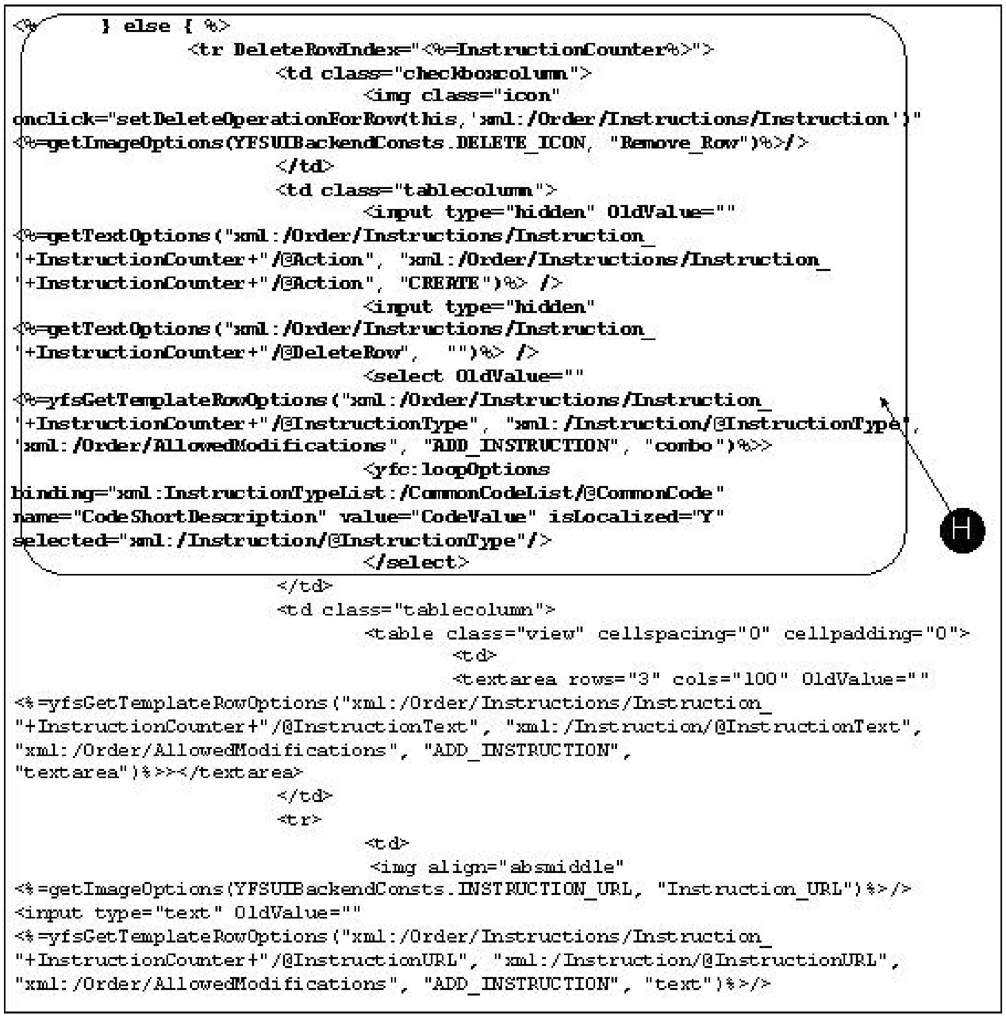
```
   <input type="hidden" id="userOperation" name="userOperation" value="" />
   <input type="hidden" id="numRowsToAdd" name="numRowsToAdd" value="" />
   <input type="checkbox" name="checkbox" value="checkbox"
onclick="doCheckAll(this);"/>
        <%}else { %>
              &nbsp;
        <%}%>
     </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"
```



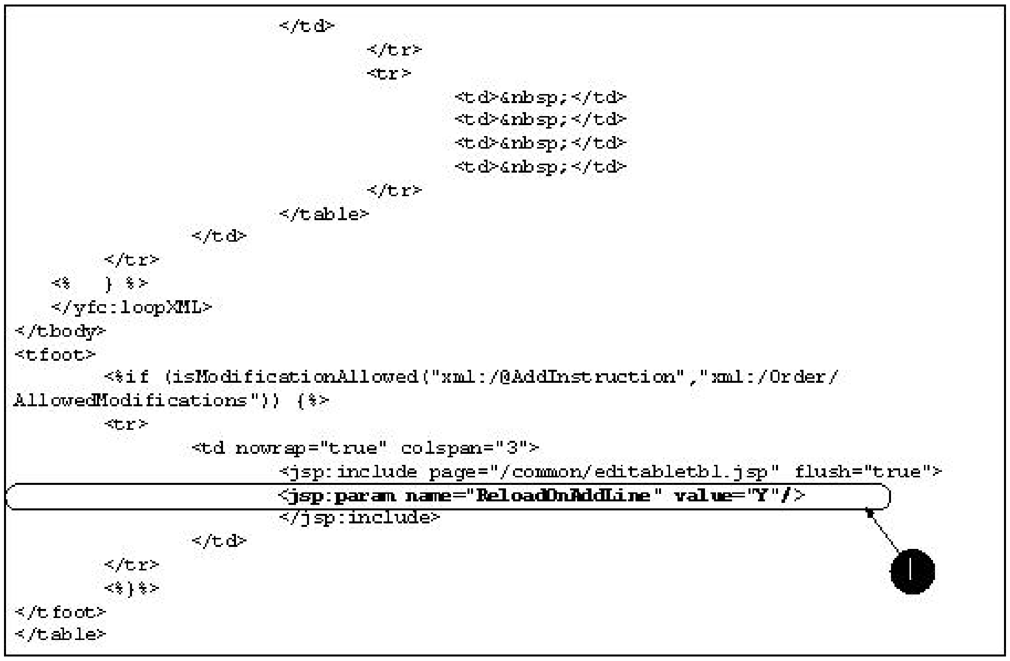
```
 value='<%=getParameter("InstructionKey")%>' name="chkEntityKey"/>
        <%}else { %>
                &nbsp;
        <%}%>
  </td>
  <td class="tablecolumn">
        <% String instTargetBinding =
"xml:/Order/Instructions/Instruction_" + InstructionCounter +
"/@InstructionType"; %>
        <select <%if(bAppendOldValue) {
%>OldValue="<%=resolveValue("xml:OrigAPIInstruction:/Instruction/
@InstructionType")%>" <%}%>
<%=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/
@InstructionText")%>"   <%}%>
<%=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>&nbsp;</td>
                <td>&nbsp;</td>
                <td>&nbsp;</td>
                <td>&nbsp;</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>
                                        <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>&nbsp;</td>
                                        <td>&nbsp;</td>
                                        <td>&nbsp;</td>
                                        <td>&nbsp;</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>
```

The following table explains the callouts.

Callout	Desc ription
C	The /webpages/yfsjspcommon/editable_util_lines.jspf file has been included in the list innerpanel.jsp. This file merges 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.
G	The OldValue attribute of each editable field is set to the value of the original API output using the map previously created. This ensures the value modified by the user is 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 row dynamically when the user clicks on the plus icon under the table, the /common/editabletbl.jsp file is included next. Pass "Y" for the ReloadOnaddLine attributes to this JSP. Each time a new row is added, the screen refreshes. Before these code changes, the new rows are added dynamically without refreshing the screen.

Adding a Lookup

Lookups enable users to select a list of options rather than typing in data. The following figure shows the available lookup icons and the fields associated with them.

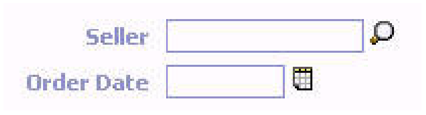


Figure 2. Lookup Icon

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

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 pop-up 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 particular type of Lookup on field, place the appropriate icon directly to the right of the Lookup.

Creating a User-Sortable Table

About this task

In any table, a user can click a column heading to sort the results. If a user clicks the same column heading again the results will sort in reverse order.

Table sorting does not create a new call to the API, it sorts the data that is displayed in the selected column.

To create a user-sortable table:

Procedure

1. Use table.htc in the style attribute for the table. If you are using the default CSS files of Sterling Selling and Fulfillment Foundation, you can use class="table" for the <table> tag.
2. The table should have <tbody> and <thead> tags that include <td> tags. 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.

   This example shows the tags needed for creating a 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="EntityK
          </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>
```

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 "About Centralized Themes."

For displaying graphs of data, the Presentation Framework uses the jbchartx.jar file from Visual JChart. Since the tool may change in a future version, you should perform your own evaluation of which charting tool to use. The Presentation Framework integrates with FusionCharts to generate charts.

Why FusionCharts

There are various advantages to integrating with FusionCharts for generating charts. Some of these are:

- Renders animated and interactive charts, as opposed to static images rendered by traditional charting components.
- Uses XML data interface that makes it cross-browser and cross-platform compatible. You can use it with any scripting language and database.
- Has a small size that makes it suitable for dynamic and interactive charting, even on narrow bandwidth connections.

- Leverages Macromedia Flash Player to make the thin client behave thick.
- Enables you to dynamically change chart types and data on the client, without invoking any server requests.

Customizing the Menu Structure

When creating customized screens, verify 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 Applications Manager 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 Selling and Fulfillment Foundation provides 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, verify 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.

Creating Custom Menus

About this task

To create a custom menu:

Procedure

1. Create a new menu using the graphical user interface described in the Sterling Selling and Fulfillment Foundation:*Configuration Guide*.
2. Open the *INSTALL_DIR*/extensions/global/resources/extnbundle.properties 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.

   Note: Ensure that the following file does not exist:

   `INSTALL_DIR/resources/extn/extnbundle.properties`

   This file must be removed because it will conflict with the extensions build process for bundle entries.

Localizing the Menu Structure

About this task

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

Note: When localizing the menu, verify 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.

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. 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 line's 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 line's JSP to be used in the order line detail screen.

Example 10–3 shows sample code for specifying screen navigation behavior.

```
 <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 in a detail view based on the parameter which is invoking the detail view.

You can call the getDetailHrefOptions() JSP function, using extra parameters that are passed to the target detail view.

The required format to pass these extra parameters should have 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 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 Hierarchy tree, the Selection Key Name should be set to this to ensure the correct key is passed to the order line detail screen.

Disabling Direct Navigation to Detail Screens

About this task

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

For example, the Order Search provided by the application 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 Hierarchy tree. 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 verify that the resource sequence is lower than the existing list view.

Chapter 11. Customizing Event Handlers in the JSP Console

About Event Handlers in the JSP Console

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 screen. For example, onblur event for input or onunload event for the page.
- Events raised by the Service Definition Framework for the page, for example before saving the data in a detail view or before invoking search in a search view.

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 numeric entry.

It is recommended that you limit client-side field-level validations to a minimum. 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 Presentation Framework function. Therefore, if you are using a numeric or date field, your function may return invalid data. You must call the Presentation Framework JavaScript utility functions to first validate the date and number before you proceed with your validations.

Screen-Level Event Handler

The Presentation Framework uses the `onload` event on document.body. All other events are available for your use. The 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="/extensions/global/webpages/scripts/om.js">
</script>
<script language=jscript>
window.document.body.attachEvent("onload", myFunc)
</script>
```

This causes the myFunc() function to run when the HTML is loaded. Note that the body of the myFunc function must exist within the *INSTALL_DIR*/extensions/global/webpages/scripts/om.js file.

The 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 *INSTALL_DIR*/extensions/global/webpages/scripts/extn.js file.

The 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="/extensions/global/webpages/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>
```

Creating Field-Level Validations

About this task

To create field-level validations:

Procedure

1. Customize the view.
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 select the Refresh Entity Cache icon.
   - If you want to update all resources - Select the Refresh Cache icon.
5. Log in to the application again to test your changes.

Creating Screen-Level Validations

About this task

To create screen-level validations:

Procedure

1. Customize the view where 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 select the Refresh Entity Cache icon.
   - If you want to update all resources - Select the Refresh Cache icon.
4. Log in to the application again to test your changes.

Chapter 12. Working with Document Types and Demand Records

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. 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, the 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), the application first tries 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.

Creating New Set of Screens for New Document Type

About this task

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

Procedure

1. From the Resource Hierarchy tree, navigate to the Order entity (resource ID order).
2. Select **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 does not conflict with any resource IDs that might ship with future releases. recommends that you choose any prefix except one that begins with the letter Y (which is reserved for use by Sterling Selling and Fulfillment Foundation).

   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 *INSTALL_DIR*/extensions/global/resources/extnbundle.properties file, and in the corresponding properties files for each locale.

   Note: Ensure that the following file does not exist:

   INSTALL_DIR/resources/extn/extnbundle.properties

This file must be removed because it will conflict with the extensions build process for bundle entries.

4. Modify the Document Type for the new resources of resource type entity to the new Document Type.
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 "Incorporating Customized Views Across the Application."

Customize the Alert Details View

About this task

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 link to also point to the main detail view of your document type, you must customize the Alert Details view.

To customize the Alert Details view:

Procedure

1. Create a new Link ID under the new detail view of Alerts.
2. Point the new Link ID to go to the newly created view of your new document type.
3. 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.
4. Change the sequence of the new Alert detail view so that the new view becomes the default Alert detail view. For details, see the *Configuration Guide*.
5. In order to make all links point appropriately to your new view, follow the steps in "Incorporating Customized Views Across the Application."
6. 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 exceptions. 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.

Working with Demand Records

Some document types have associated views that enable you to view demands created. Document types are not associated with creating supply.

Sterling Selling and Fulfillment Foundation provides consoles for viewing the demands stemming from Sales Orders, Returns, and Purchase Order (document type 0001, 0003, and 0005 respectively) 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.

Extending Demand List View for Document Types

About this task

To extended demand list view for document types:

Procedure

1. Extend the detail view (YIMD215) by following the directions in "Customizing Detail Views in the JSP Console."
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 the view defined by Sterling Selling and Fulfillment Foundation.

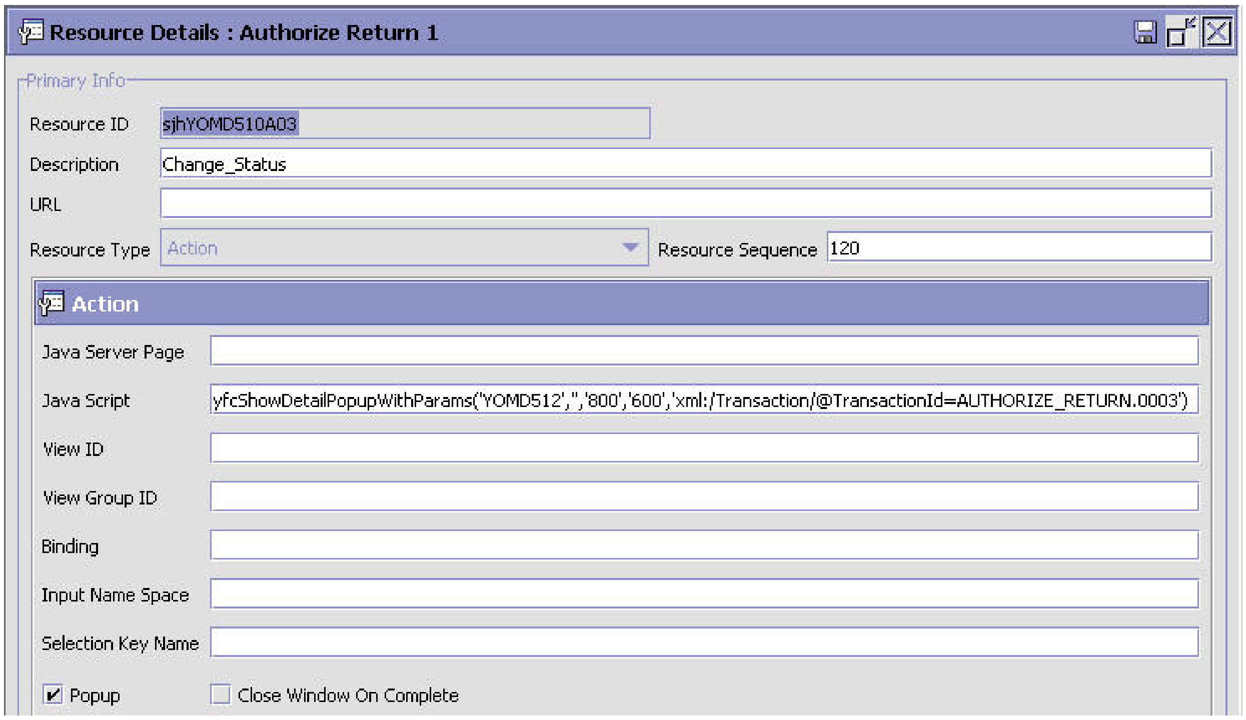
Chapter 13. Actions, XML Binding, APIs, Dynamic Namespaces, and Credit Card Numbers

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 Process Modeling of the Applications Manager 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.

The figure shows the Status Change Action for Authorizing Returns screen.


Resource Details : Authorize Return 1

Primary Info

Resource ID	sjhYOMD510A03
Description	Change_Status
URL	
Resource Type	Action
Resource Sequence	120

Action

Java Server Page	
Java Script	yfcShowDetailPopupWithParams('YOMD512','','800','600','xml:/Transaction/@TransactionId=AUTHORIZE_RETURN.0003')
View ID	
View Group ID	
Binding	
Input Name Space	
Selection Key Name	

☑ Popup ☐ Close Window On Complete

Create an Action from an Inner Panel

About this task

To create an action from an inner panel:

Procedure

1. From the tree in the application rules side panel, choose Process Modeling. Configure a new transaction in the pipeline, along with the pickup 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."
3. Verify that the new transaction ID (created in Step 1) and the view resource ID are passed as input to the JavaScript function.

For This Entity...	Use This Resource ID...
Shipment	YOMD770
Load	YDMD280
Order	YOMD390
Purchase Order	YOMD3390
Returns	YOMD512

XML Binding

The input and output of APIs is in the form of an XML document. Using XML documents enables the Presentation Framework to provide an XML binding mechanism where 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 onto 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 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: This example does not show all of the custom attributes returned by the JSP functions. It only shows the ones relevant to this topic.

In another example, the user enters `Telephone` in the input box. When the data is posted to the server, the 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 Presentation Framework parses the binding string to form the XML, the binding string must follow the syntax below.

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

  The various XML binding parameters used are as follows:
  - **xml:—**used literally
  - **Namespace—**namespace containing the XML to which this binding applies. If not specified, it is taken 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.

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>
```

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 are not posted to the API.

To uniquely identify each input as 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 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: When this data is posted, all XML bindings containing the same special postfix are combined into the same XML element in the API input.

```
xml:/Order/OrderLines/OrderLine_1/@ShipNode
```

and

```
xml:/Order/OrderLines/OrderLine_2/@ShipNode
```

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. 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")%>/>
```

Passing Data to APIs

When customizing the user interface, you must verify 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.

The application provides 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 is any detail screen with 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 Hierarchy tree, the Input Namespace field in the action resource should be set appropriately to verify that the correct data from the console is passed to the save API. See the Sterling Selling and Fulfillment Foundation: *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 invoked on the user interface, an XML is formed from all of the input fields containing the namespace specified for that action in the Resource Hierarchy tree. This XML is then passed to the API configured under that save action. For more information, see "XML Binding."

Another place where input namespace is frequently used is for user interface screens that have search criteria. The search criteria is passed to list APIs to fetch the 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 runs 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, no other steps are 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 Hierarchy tree 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 to retrieve all the data that 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 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 an 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."

Available Dynamic Attribute Namespaces

Sterling Selling and Fulfillment Foundation has 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 an 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 "Passing Data to APIs."

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 typically this code immediately follows the include statements located at the beginning of a JSP.

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 Presentation Framework has two APIs, getTextOptions() and yfsGetTextOptions() , that permit you not to 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 definitions and mappings are contained in two files:

- yfsdatatypemap.xml file—maps abstract data types to XML attribute names.
- datatypes.xml file—defines data types.

Abstract Data Type Mappings

XML attributes are mapped to abstract data types. The mappings are contained in the *INSTALL_DIR*/repository/xapi/template/merged/resource/ 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" />
```

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 *INSTALL_DIR*/repository/datatypes/datatypes.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">
   <XMLType Type="QUANTITY"/>
</DataType>
```

For a definition of the standard abstract data types, see the *INSTALL_DIR*/ repository/datatypes/datatypes.xml file.

For a list of attributes supported datatypes.xml , see "Data Type Reference."

Data Type Determination

Sterling Selling and Fulfillment Foundation uses the yfsdatatypemap.xml file for data type determination.

First the application searches the file to find the string specified for binding (including the xml: prefix). If the entire binding string is not found, the application searches 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 datatypes.xml file, the Presentation Framework API forms an appropriate HTML string with custom attributes that are then used on the client side.

Data Type Validation

A JavaScript function on the client side runs 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.

Displaying Credit Card Numbers

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.
- 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:/PaymentMethod/
@DisplayCreditCardNo"))%>
            </a>
        </td>
 <% } else { %>
      <td class="protectedtext">

       <%=showEncryptedCreditCardNo(resolveValue("xml:/PaymentMethod/
@DisplayCreditCardNo"))%>&nbsp;
```

```
           <yfc:getXMLValue binding="xml:/PaymentMethod/
@DisplayCreditCardNo"/>&nbsp;
       </td>
 <% } %>
```

- Then create a pop-up window that opens when the hyperlink is clicked.
- Call getDecryptedCreditCardNumber() in the pop-up 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 Applications Manager, 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:

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

And specify the Template field according to the following example:

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

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 pop-up window that displays a specified credit card number in a decrypted format.

Chapter 14. User Interface Style Reference

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 the following table.

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.
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.

Control and Tag	Available Classes	Description
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.
	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.

Page Layout

The following table describes the guidelines to follow when customizing screens in the Application Console.

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 pop-up 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 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.
Labels and Inputs	Specific names for labels on the screen should be the same as the names used in the Applications Manager. If not used in the Applications Manager, 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 pop-up windows.
Page Title-bars	Should describe the current page and contain a list of available views when there is more than one view available.

Object	Standards
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.

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.

Chapter 15. Programming Standards for the JSP Console Interface

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.

Valid HTML Tags and Attributes

Follow this HTML reference material to help guide you in using the HTML attributes as they are used by the application .

Note: You can also use any other HTML attributes, as long as you devise your own set of standards.

The following table lists the recommended standard HTML tags and their attributes. For each HTML tag, use only the attributes listed in the Tag Attribute column.

HTML Tag	Tag Attributes
<% %>	keep at the top of the JSP wherever possible
<a>	href
<imp>	alt
	border
	name
	src
	style

HTML Tag	Tag Attributes
<input>	class
	maxlength
	name
	onblur
	style
	value
<option>	binding
	selected
	type
	value
<select>	class
	name
<table>	editable
	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)

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

Conventions for Naming Controls

When using the Presentation Framework function for XML binding input controls in your JSP, do not set the name attribute for that control. In other words, avoid naming each control. Instead, access the control through the HTML object or DOM hierarchy. If you want to name a control, ensure that the name is unique within each page.

Internationalization

The Presentation Framework provides the ability to write an internationalized application. To enable this, it provides 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

Validating Your HTML and CCS Files

You can validate both 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 alternative, after you finish coding a form, you can open it in any visual HTML editor to validate that the HTML is well-formed.

Chapter 16. CSS Theme File Reference

CSS Themes for the JSP Console

The standardSterling Selling and Fulfillment Foundation theme uses the Tahoma font as specified within the CSS files. If you use a different sized font, you may encounter display problems, such as truncation of the drop-down list items. In such situations, you can edit the CSS file and specify properties that enable the screen to display correctly. Use the classes and properties described in the following table.

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.
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.

Class	Description
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.
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.
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.
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.

Class	Description
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.

Chapter 17. JSP Functions for the Console JSP Interface

changeSessionLocale

Description

While locale is configured at the user level, you can also dynamically switch to a specific locale by using the changeSessionLocale JSP function.

Syntax

void changeSessionLocale(String localecode)

Input Parameters

localecode - Locale you want to switch to.

equals

Description

The equals JSP function is a cover over Java's equal function that handles objects that are null, contain zero, or more white spaces. In such 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");
%>
```

getCheckBoxOptions

Description

This JSP function is a standard function to XML bind checkboxes when modification rules need not 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>
```

getColor

Description

This JSP function returns the HTML color in hexadecimal code based on the specific color object.

Syntax

public String getColor(java.awt.Color color)

Input Parameters

color - Required. Color object.

getComboOptions

Description

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 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>
```

getComboText

Description

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 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)%>
```

getDateOrTimePart

Description

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));
```

getDateValue

Description

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>
```

getDBString

Description

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"))%>
```

getDetailHrefOptions

Description

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 Hierarchy tree. 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 an <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>
```

getDetailHrefOptions (with additional parameter)

Description

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 an <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 -
```

getDoubleFromLocalizedString

Description

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.

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>
<%}%>
```

getElement

Description

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 a part of theSterling Selling and Fulfillment Foundation DOM utility package. To see the APIs available in this package, refer to the 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"));
                       }
%>
```

getImageOptions

Description

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 /smcfs/console/icons, the image file is first searched for inside /extensions/global/webpages/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 /smcfs/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.

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 ("smcfs/console/icons/shipnode.jpg"
"Search_for_Organization") %> />
```

getLocale

Description

This JSP function returns a YFCLocale object that represents the locale of the user logged in to Sterling Selling and Fulfillment Foundation.

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.

```
<%
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/
@TotalInternalUnassignedDemand" />">
   <tr>
     <td style="height:10px;width:15px"></td>
   </tr>
 </table>
<%}%>
```

getLocalizedStringFromDouble

Description

This JSP function returns a representation of a string value and displays it in the correct format for a specific locale.

Sterling Selling and Fulfillment Foundation always displays 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.

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);
```

getLocalizedStringFromInt

Description

This JSP function returns a representation of an integer value and displays it in the correct format for a specific locale.

Sterling Selling and Fulfillment Foundation always displays 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>
```

getLoopingElementList

Description

This JSP function can be used as an alternative to the loopXML JSP tag.

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 is 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>
```

getNumericValue

Description

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 the Sterling Selling and Fulfillment Foundation 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>
```

getParameter

Description

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 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>
```

getRadioOptions

Description

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.

Note: If only the name parameter is passed, the value of the value parameter defaults to the same value passed to the name parameter.

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>
```

getRequestDOM

Description

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"/>
```

Output of getRequestDOM is:

```
<root>
  <Order OrderNo="Order0001" OrderType="Customer"/>
  <User UserName="user01"/>
</root>
```

getSearchCriteriaValueWithDefaulting

Description

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 is 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>
```

getTextAreaOptions

Description

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")%>
```

getTextOptions

Description

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 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")%>/>
```

getUITableSize

Description

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")%>"
```

getValue

Description

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 explains 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");
%>
```

goToDetailView

Description

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 is shown to the user.

Syntax

void goToDetailView(HttpServletResponse response, 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 is 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 is 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);
%>
```

isModificationAllowed

Description

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>
 <%}%>
```

isPopupWindow

Description

This JSP function determines whether or not the current window is displayed in a pop-up window. Use this function when the logic in your screen must differ when it appears in a pop-up window.

Syntax

isPopupWindow()

Input Parameters

None.

Output Parameters

A boolean indicating whether or not the current window is displayed in a pop-up 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 pop-up 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>
```

isTrue

Description

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");
```

isVoid

Description

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>
<%}%>
```

resolveValue

Description

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");
%>
```

showEncryptedCreditCardNo

Description

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/@DisplayCreditCardNo"))%>
```

userHasOverridePermissions

Description

This JSP function determines whether or not the current login user has permission to override the modifications rules configuration.

Syntax

boolean userHasOverridePermissions()

yfsGetCheckBoxOptions

Description

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 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/@ChargeActualFreightFlag","Y","xml:/Order/AllowedModifications")
%>/>
```

yfsGetComboOptions

Description

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 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>
```

yfsGetImageOptions

Description

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.

A Java constants file keeps the image path and icon centralized. If the path starts with /smcfs/console/icons, the image file is first searched for inside /extensions/global/webpages/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 /smcfs/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")%>/>
```

yfsGetTemplateRowOptions

Description

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">&nbsp;</td>
    <td class="tablecolumn">&nbsp;</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">&nbsp;</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">&nbsp;</td>
        <td class="tablecolumn">&nbsp;</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>
```

yfsGetTextAreaOptions

Description

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 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>
```

yfsGetTextOptions

Description

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 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")%>/>
```

Chapter 18. JSP Tag Library for the Console JSP Interface

callApi

Description

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 Hierarchy tree, 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.

callAPI (Alternative Method)

Description

The callAPI JSP tag also supports a way to call APIs within JSPs without defining the API in the Resource Hierarchy tree. 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 is called. When using this alternative method of callAPI either apiName or serviceName is required.

suppressInputDecode - Optional. Set the value of this attribute to true, if you want to suppress the recursive html decoding of all the attributes of inputElement as well as all the child elements.

serviceName - Optional. The name of the service (from Service Definition Framework) that is 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 is 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 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 "Passing Data to APIs."

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 Hierarchy tree. 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'/>
```

getXMLValue

Description

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>
```

getXMLValueI18NDB

Description

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>
```

hasXMLNode

Description

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>
```

i18n

Description

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 Selling and Fulfillment Foundation: *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>
```

i18ndb

Description

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 Selling and Fulfillment Foundation: *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>
```

loopOptions

Description

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 Selling and Fulfillment Foundation searches 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 the application searches 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 ensures that data entered by the end user is 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>
```

loopXML

Description

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 "JSP Functions for the Console JSP Interface."

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")%>',
'<%=resolveValue("xml:item:/Item/PrimaryInformation/@DefaultProductClass")
%>','<%=resolveValue("xml:item:/Item/@UnitOfMeasure")%>')"
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>
```

makeXMLInput

Description

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.

<indexterm>makeXMLInput JSP tag;JSP tag library:makeXMLInput</indexterm>

```
<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:getXMLValue
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>
```

makeXMLKey

Description

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."

Chapter 19. JavaScript Functions

About JavaScript Functions for the Console JSP Interface

TheSterling Selling and Fulfillment Foundation UI uses JavaScript functions to perform client-side operations such as opening pop-up 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 the JavaScript functions supplied by the application's UI layer.

Note that the application also uses 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 the UI infrastructure does not provide JavaScript functions, you can write and use your own as needed.

Lookup

callLookup. Uses [GET]

invokeCalendar. Uses [GET]

yfcShowSearchPopup. Uses [GET]

Control Name

ignoreChangeNames

yfcDoNotPromptForChanges

yfcDoNotPromptForChangesForActions

yfcHasControlChanged

yfcSetControlAsUnchanged

yfcSpecialChangeNames

Event Handler

validateControlValues

yfcBodyOnLoad

yfcGetSaveSearchHandle

yfcGetSearchHandle

yfcValidateMandatoryNodes

Show Detail

showDetailFor. Uses [GET]

showPopupDetailFor. Uses [GET]

yfcChangeDetailView. Uses [POST]

yfcShowDefaultDetailPopupForEntity. Uses [GET]

yfcShowDetailPopupWithDynamicKey

yfcShowDetailPopupWithKeys. Uses [GET]

yfcShowDetailPopupWithParams

Show List Pop-up

yfcShowListPopupWithParams. Uses [GET]

Other

doCheckAll

doCheckFirstLevel

expandCollapseDetails

getAttributeNameFromBinding

getCurrentSearchViewId

getCurrentViewId

getObjectByAttrName

goToURL

showHelp. Uses [GET]

yfcAllowSingleSelection

yfcDisplayOnlySelectedLines

setRetrievedRecordCount

callLookup

Description

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") %> />
```

doCheckAll

Description

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>
```

doCheckFirstLevel

Description

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 that 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/
AdditionalServiceItems/Item/@ItemID")%>"><yfc:i18n>Item_ID</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/@UnitOfMeasure")%>"><yfc:i18n>UOM</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/PrimaryInformation/@Description")%>">
<yfc:i18n>Item_Description</yfc:i18n></td>
         <td class="tablecolumnheader"
style="width:<%=getUITableSize("xml:AdditionalServiceItem:/OrderLine/
AdditionalServiceItems/Item/@Price")%>"><yfc:i18n>Price</yfc:i18n></td>
     </tr>
   </thead>
```

expandCollapseDetails

Description

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:/ChargeTransactionDetail/@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:/
ChargeTransactionDetail/@CreditAmount")%>">
```

```
               <yfc:getXMLValue binding="xml:/ChargeTransactionDetail/
@CreditAmount"/>
           </td>
        </tr>
   </yfc:loopXML>
</tbody>
```

getAttributeNameFromBinding

Description

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.

getCurrentSearchViewId

Description

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.

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>
```

getCurrentViewId

Description

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:/
InventoryInformation/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>
```

getObjectByAttrName

Description

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" or "yfsGetComboOptions".

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;
     }
   }
 }
```

getParentObject

Description

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 run 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;
}
```

goToURL

Description

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>
```

ignoreChangeNames

Description

Whenever any detail view is posted, the 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 Presentation Framework checks for this custom property. If the property is set, the automatic name changing does not happen.

The 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 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().

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>
```

invokeCalendar

Description

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:
/InventoryInformation/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>
```

invokeTimeLookup

Description

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>
```

showDetailFor

Description

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 pop-up 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>
```

showDetailForViewGroupId

Description

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 pop-up 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>
```

showHelp

Description

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 typically want only the overall system help and not the screen-level context sensitive help.

showPopupDetailFor

Description

This JavaScript function shows the default view of the current entity in a pop-up 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 pop-up window.

name - Required. Pass as blank space (" "). Not used.

width - Required. Horizontal size of the pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up window. When you select the transaction date, if the current screen is a pop-up window, another pop-up 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>
```

validateControlValues

Description

This JavaScript function checks for client-side validation errors. When the user enters invalid data in an input field, the 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:/InventoryInformation/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>
```

yfcAllowSingleSelection

Description

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))
  {
     ...
   }
 }
```

yfcBodyOnLoad

Description

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 Presentation Framework take it.

```
function window.onload()     {
    if (!yfcBodyOnLoad()
&& (!document.all('YFCDetailError'))) {
     return;
    }
//Do your special processing here
}
```

yfcChangeDetailView

Description

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>
```

yfcChangeListView

Description

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.

Note: Use the *INSTALL_DIR*/properties/customer_overrides.properties file to set the yfs.ui.MaxRecords property.

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.

yfcDisplayOnlySelectedLines

Description

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>
```

yfcDoNotPromptForChanges

Description

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 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.

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>
```

yfcDoNotPromptForChangesForActions

Description

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 Application Console 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 is displayed by default. Call this JavaScript method to avoid this validation. You can call this method for 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>
```

yfcGetCurrentStyleSheet

Description

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();
```

yfcGetSaveSearchHandle

Description

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."

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);
...
```

yfcGetSearchHandle

Description

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."

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);
...
```

yfcHasControlChanged

Description

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 pop-up 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;
            }
       }
    }
}
```

yfcMultiSelectToSingleAPIOnAction

Description

This JavaScript function replaces the yfcMultiSelectToSingleAPI() JavaScript function, which has been deprecated.

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 run while multiple selections have been made by the user, the API runs once for each selected record. This function enables you to configure an action that calls an API that runs 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 elements 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.

yfcSetControlAsUnchanged

Description

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."

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 the data on screen will be lost" 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."

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>
```

yfcShowDefaultDetailPopupForEntity

Description

This JavaScript function shows the default detail view of an entity in a pop-up 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"
```

yfcShowDetailPopup

Description

This JavaScript function shows a specific view ID in a pop-up 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 pop-up window. If passed as an empty string, the pop-up 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 pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up window.

Return Values

None.

Example

This example shows how the Modification Reason Code pop-up 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);
    }
}
```

yfcShowDetailPopupWithDynamicKey

Description

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 pop-up 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 pop-up window. If passed as an empty string, the default detail view of the specified entity is shown in the pop-up 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 pop-up 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());
}
```

yfcShowDetailPopupWithKeys

Description

This JavaScript functions shows a specific view ID in a pop-up 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 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 pop-up 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 pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up 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 pop-up 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 pop-up window is invoked. If this name is not passed (or is passed as null), the check is not performed, and the pop-up 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);
}
```

yfcShowDetailPopupWithKeysAndParams

Description

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 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 pop-up 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 pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up 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 pop-up 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 pop-up window is invoked. If this name is not passed (or is passed as null), the check is not performed, and the pop-up 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=process')
```

yfcShowDetailPopupWithParams

Description

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 pop-up 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 pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up 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 pop-up window is displayed using this function. The notes pop-up 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);
```

yfcShowListPopupWithParams

Description

This JavaScript function shows a specified list view in a pop-up 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 pop-up 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 pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up 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/@UnitOfMeasure="+UOM;
      sAddnParams=sAddnParams + "&xml:/InventoryAudit/@ProductClass="+PC+"&xml:/InventoryAudit/@Or

       yfcShowListPopupWithParams(sViewID,"",'900', '500','',entity,
sAddnParams);
}
```

yfcShowSearchPopup

Description

This function invokes the specified search view in a pop-up 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 pop-up 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 pop-up window. Measured in pixels. If passed as 0, a certain default width is used.

height - Required. Vertical size of the pop-up 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 pop-up 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 pop-up window.

From the search pop-up 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);
}
```

yfcSpecialChangeNames

Description

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.

yfcSplitLine

Description

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.

The following table lists the attributes to use at each cell level for determining the behavior of the newly created rows.

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 "*_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.
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.jpg"
          <% 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" NewRese
<%=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" ewClassName="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"
<%=getTextOptions("xml:/InspectOrder/ReceiptLines/FromReceiptLine_" +
ReceiptLineCounter + "/ToReceiptLines/ToReceiptLine_1/@InspectionComments",
"")%>/>
    </td>
  </tr>
 </yfc:loopXML>
</tbody>
```

yfcValidateMandatoryNodes

Description

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>">
```

yfcFindErrorsOnPage

Description

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;
}
```

setRetrievedRecordCount

Description

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 displays `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 automatically displays 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>
```

Chapter 20. Data Type Reference

Data Type Reference for the Console JSP Interface

The DataType node contains UIType and XMLType nodes. For the Presentation Framework, the attributes specified in UIType override those specified in XMLType, which in turn override those specified in DataType.

The following table lists which nodes are supported by specific 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 default value specified in the yfs.install.displaydoublequantity property in the yfs.properties file. **Note:** To modify this property, add an entry for it in the *INSTALL_DIR*/properties/ customer_overrides.properties file. For additional information about modifying properties and the customer_overrides.properties file, see the Sterling Selling and Fulfillment Foundation: *Properties Guide*.	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

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

Notices

Trademarks

Product Number: xxxx-xxx

Printed in USA