Selling and Fulfillment Foundation

Customizing the Web UI Framework

Release 9.0

March 2010

Sterling Commerce
An IBM Company

Contents

Extensibility in the Web UI Framework

Extensibility in the Web UI Framework for Custom Developers

Extensibility allows you to customize the user interface of an existing out-of-the-box installation of the application using the Extensibility Workbench. It allows you to customize the existing installation at runtime without recompiling or changing the original source code.

Web UI Framework extensibility also allows you to modify the Struts, non-XAPI mashup, and XAPI mashup layers. If you do not use the Web UI Framework, the XAPI mashup layer is not available.

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

With extensibility, you can open an existing screen and bring up the same user interface tools that were used by application developers to build the screen. You can add controls (like buttons, labels, and grid columns), panels, data sources, and other items. The Extensibility Workbench allows you to personalize and localize the application. It helps you display more relevant and organized data.

Controls and panels are also known as widgets. The Extensibility Workbench allows you to add new widgets to a UI screen, override default field labels, and customize themes. For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

The Extensibility Workbench allows you to remove widgets that were added through extensibility. However, you cannot remove any widgets that were present in the base/original screen. Also, you can hide (not remove) an existing widget.

The Extensibility Workbench also allows you to change the properties of widgets. However, property changes are limited to a particular set of properties. This ensures that arbitrary properties are not allowed to change, resulting in upgrade issues.

Use the Extensibility Workbench to make changes to parts of a screen. If you want to customize an existing installation with new screens, use the Designer Workbench, which you access from a link in the Extensibility Workbench. These workbenches let you work with the two kinds of extensibility:

- Differential extensibility, where you use extensions to add overlays onto a base screen. The addition is the difference between the final screen (after adding the extensions) and the out-of-the-box screen (without the extensions). Also, you can use differential extensibility to re-arrange the components of a screen.

  You can use the Extensibility Workbench for differential extensibility.

- Override extensibility, where you replace the out-of-the-box screens with new screens.

  You can use the Designer Workbench for override extensibility.

Differential Extensibility in the Web UI Framework

With differential extensibility, you can customize parts of a screen. Changes are overlaid on top of the base screen. Differential extensibility contrasts with override extensibility, where the entire screen is replaced.

In differential extensibility, the extensions are stored in a file that is separate from the file of the screen being viewed or edited. During runtime of the application, the extensions are applied to the functionality of the application. This kind of extensibility gives you flexibility with upgrades.

In differential extensibility, in general, you can do the following:

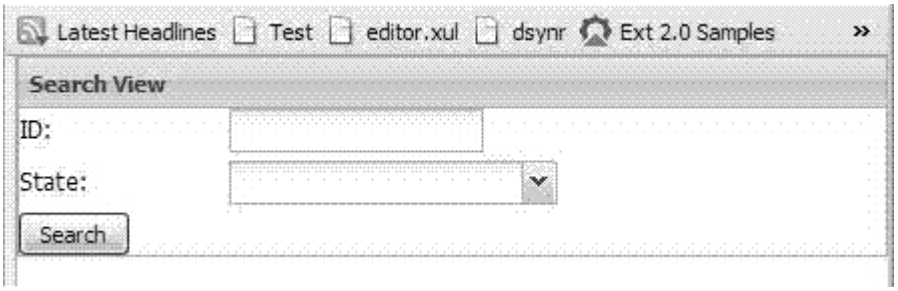
- Add new UI components to an out-of-the-box screen.
- Change an existing component such as styles, labels, and layout parameters.
- Hide a component present in an out-of-the-box screen.
- Remove a component that was added via extensibility.

Note: None of the out-of-the-box components or component properties can be deleted.

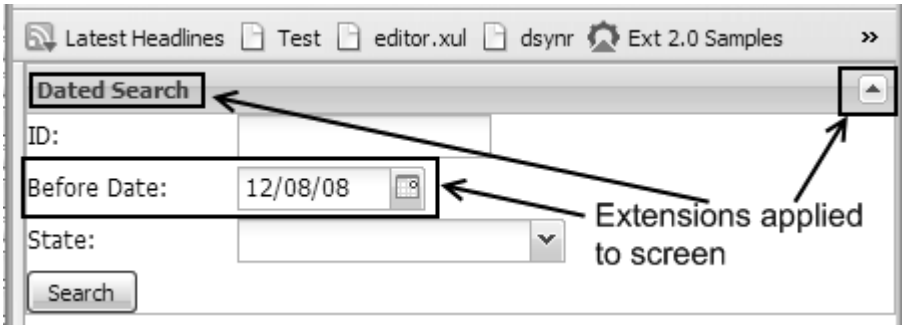
- Respond to events.

The following images show an example of an out-of-the-box screen and an extended screen. Differential extensibility was used to add a search button and a Before Date field.

Out-of-the-box screen (without extensions):



Extended screen:



Override Extensibility in the Web UI Framework

With override extensibility, you can customize a screen by completely replacing it. Override extensibility contrasts with differential extensibility, in which only parts of the screen are replaced. Use the Designer Workbench to apply override extensibility.

Note: Although you can apply override extensibility using the Designer Workbench, you are limited in the changes that you can make. Please contact Sterling Commerce Customer Support for assistance when applying override extensibility using the Designer Workbench.

Differential Extensibility Versus Override Extensibility in the Web UI Framework

Area	Differential Extensibility	Override Extensibility
Scope of Changes	Individual screen components.	Entire screen.
Screen Actions	• Add component • Change component • Hide a component present in an out-of-the-box screen • Remove a component or a component property added using the Extensibility Workbench • Override default field labels • Customize theme	Replace entire screen.
Recommended Usage	When screen requires minor enhancements with little behavioral change.	When screen requires complex enhancements, such as removing controls or changed business use cases. After an upgrade, if you are not interested in the enhancements in an out-of-the-box screen, this method is recommended.
Tool	Extensibility Workbench	Designer Workbench
Runtime Application	Extensions are applied to the functionality of the application.	Functionality that is extended is completely replaced. Both the functionality and the UI layout and/or appearance are replaced (if required).

Area	Differential Extensibility	Override Extensibility
Upgrade Issues	Increases upgrade flexibility, because original screen does not change, making individual extensions to the screen easier to apply. An extensible screen adheres to the extensibility guidelines, such as unique IDs. These IDs should not be absent from the upgraded screens.	Might increase upgrade time, especially if there are upgrade-related source code changes in the application that relate to overridden screen. The added code for XAPI mashups that are used for a new screen (if any) would be affected with the changes in the database tables or source code.
Extensible Layers	• Presentation layer (UI) • XAPI mashup layer	• Presentation layer (UI) • XAPI mashup layer • Non-XAPI mashup layer • Struts layer
Screen File Management	Extensions are stored in a different file from the files of the screen being extended. This different file must be a new JavaScript file that must be created by the user. This new JavaScript file must be included in the application using JSB definitions. • Base screens extend the sc.plat.ui.ExtensibleScreen class. These screen definitions have an identifier that is unique across the application. • Screen extensions extend the sc.plat.ui.Extension class. These extensions are registered with the Web UI Framework extension registry for the base screen's identifier.	Screen files are completely replaced. New Java Server Page (JSP) files override base JSP files. These JSP files can be designed in the Designer Workbench or from another source.

What Can Be Customized and Extended with the Web UI Framework

You can use the Web UI Framework to customize and extend any screen of the application that also follows these guidelines:

- Any extensible UI content is served to the client using a JSP (Java Server page).
- A unique identifier must be created for every screen class (the className property) and screen component (the sciId property).

  If this guideline is not followed, a console warning will alert you that more than one screen or screen component has the same ID. You can still launch and deploy the application out-of-the-box with duplicate IDs for screen or screen components, but duplicate IDs are likely to cause problems when you try to extend.
- In differential extensibility (where only parts of the screen change), the extensions are defined in an extension file which must be included with the out-of-the-box screen.
- It uses the Ext JS JavaScript framework.
- The screen class must extend from the class sc.plat.ui.ExtensibleScreen.

- It does not add controls dynamically. These controls cannot be changed by screen extensions. Also, all layouts do not support the addition of dynamic controls.

You can extend screens that were not originally created using the Web UI Framework tools (for example, hand-coded screens). However, if a screen was designed using the Web UI Framework tools, it can be easily extended because it conforms to the Web UI Framework standards.

Customizing with the Web UI Framework

The Web UI Framework allows you to plug in customizations of tasks like authentication and authorization. When you customize the application, you need to write special program code that works with the interface contracts of the default installation of the application.

You can use interface contracts to customize the following tasks:

- Authentication
- Post authentication
- Authorization
- Mashup layer
- Transaction management
- State management
- Localization
- Data type handling

Customizations also use the web.xml file and the install3rdParty tool.

Extending Versus Customizing an Ext JS Widget/Component with the Web UI Framework

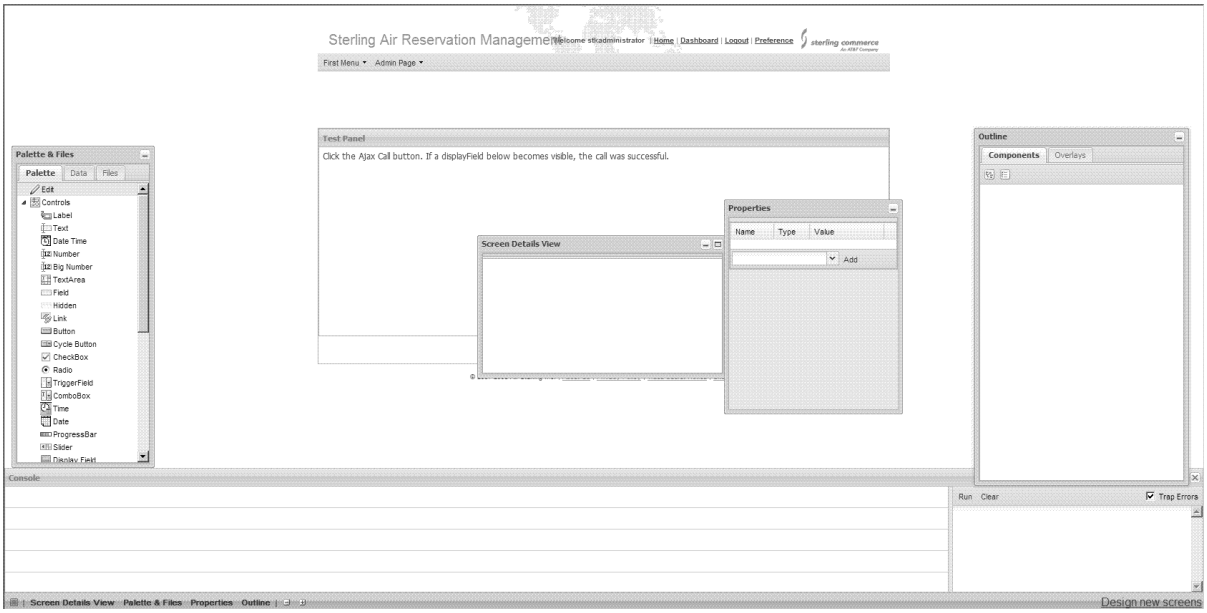
Extending and customizing are very similar. Both involve changes to the default, out-of-the-box version of the application.

Extending is a type of customization that involves creating changes in a separate file that are applied to the application, making these changes easy to identify and easy to remove.

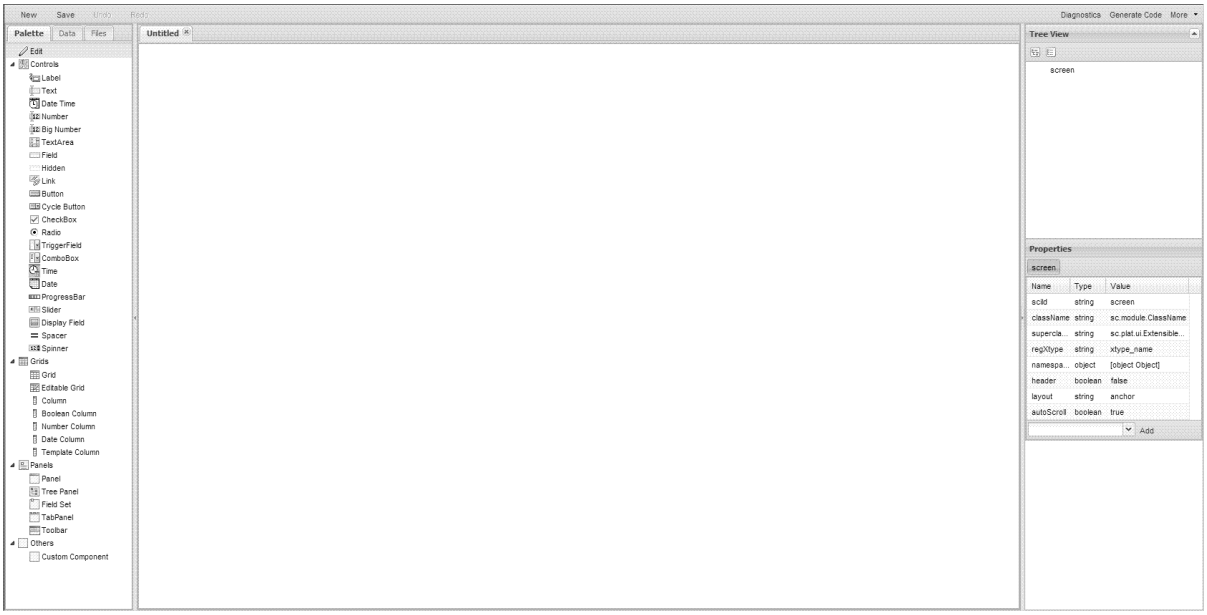
Customization can also involve more direct changes to the application that change (and do not preserve) the original configuration of the application.

Extensibility Workbench Versus Designer Workbench in the Web UI Framework

The Extensibility Workbench is used to modify the UI while the application is live and running online. The changes can be saved and later applied to the application EAR, if required. Use the Extensibility Workbench to make changes to part of the UI. To create new screens while working in the Extensibility Workbench, access the Designer Workbench through the **Design new screens** link at the bottom of the Extensibility Workbench.



The Designer Workbench is also used by application developers to first create offline the screens that you can modify using the Extensibility Workbench.



Extensibility Workbench of the Web UI Framework for Custom Developers

The Extensibility Workbench allows you to use WYSIWYG tools in an existing application to put overlays on a screen's user interface configuration.

The Extensibility Workbench allows the editing of out-of-the-box components on an existing screen.

The Extensibility Workbench includes the following components:

- Extensibility Workbench

  Used to extend an out-of-the-box screen. Access this tool from within the application by clicking **Shift + space bar**. You can also turn off the Extensibility Workbench by clicking **Shift + space bar**.

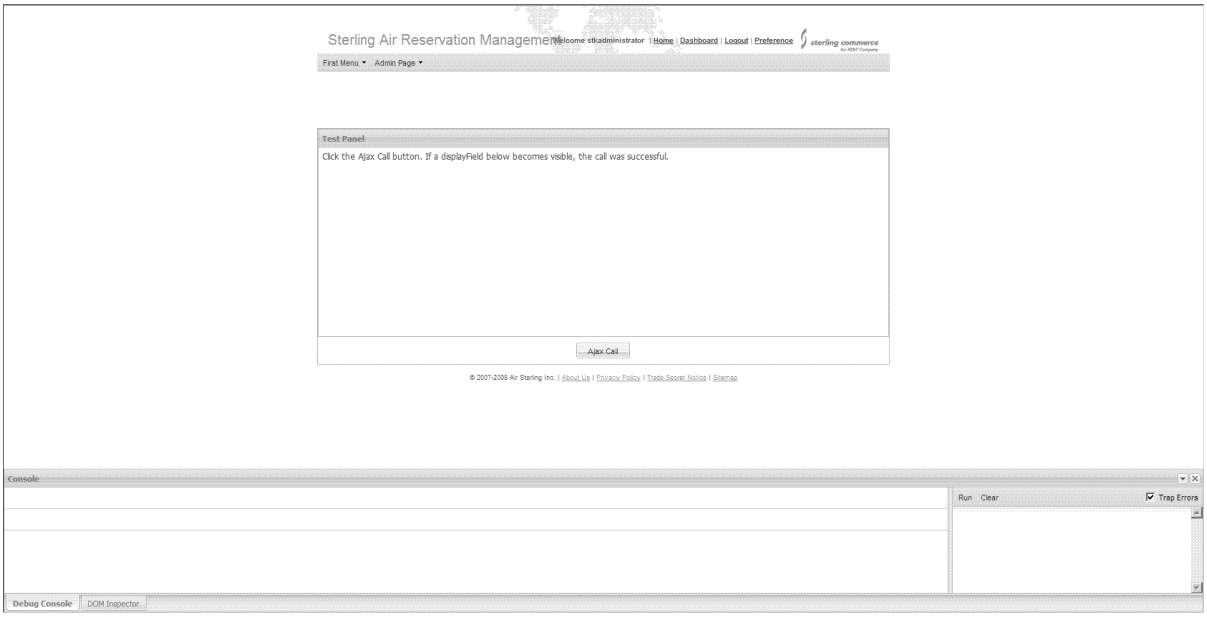
- Designer Workbench

  Used to design new screens. Access this tool from within the Extensibility Workbench by clicking the **Design new screens** link.

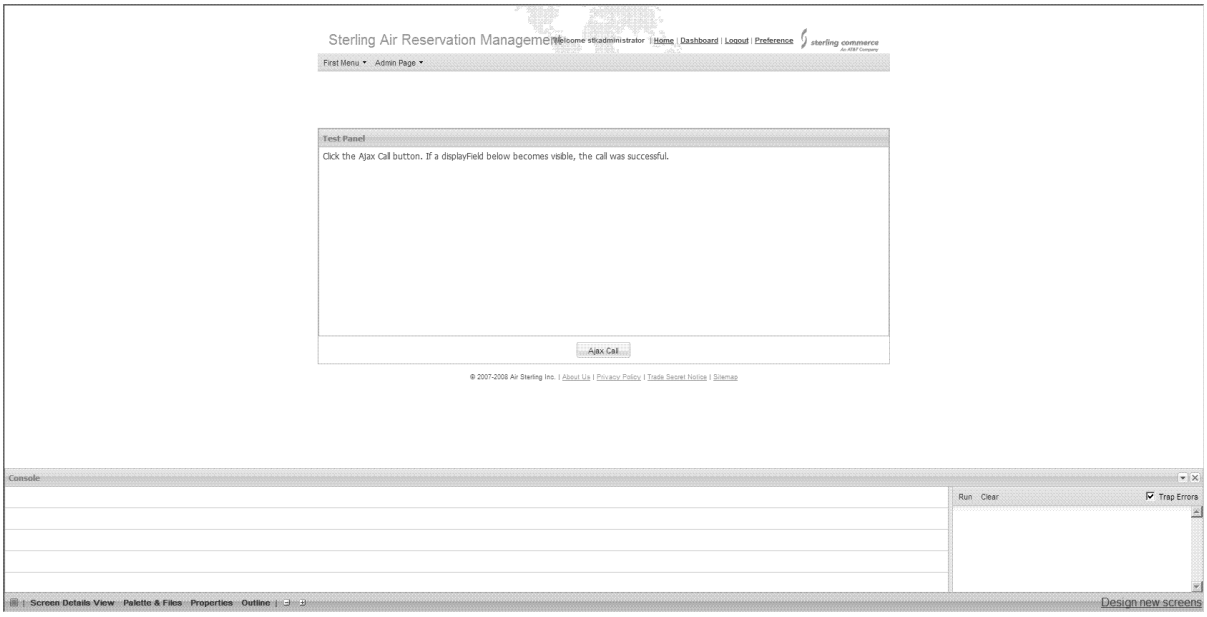
  After you access the Designer Workbench from the Extensibility Workbench, you can use the **Back** button to return to the application. However, you will need to re-activate the Extensibility Workbench.

Screens designed in Designer Workbench have to be deployed and run in an existing installation of the application to see the functional behavior. In the Extensibility Workbench, extensions are added to a screen in a live application. If a change is made to a screen, the changes can be viewed instantly. To extend a screen using the Extensibility Workbench, you have to navigate to the corresponding screen and then start extending it.

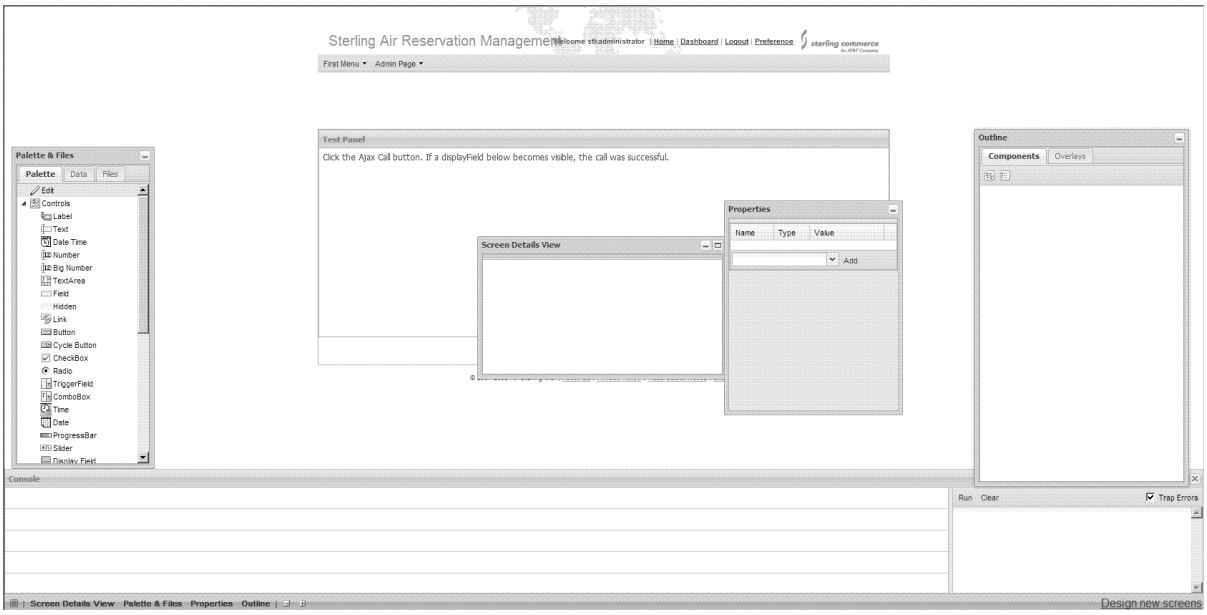
Application without Extensibility Workbench:



Application with Extensibility Workbench after clicking **Shift + space bar** (tabs and link at bottom of screen):



Application after maximizing Extensibility Workbench views by clicking plus (+) sign at bottom of screen:



Configuring the Web UI Framework Extensibility Workbench

If you are using Mozilla Firefox and the Mapping Preferences dialog box appears when you open the Extensibility Workbench, you need to configure the application to access the supporting files for the Extensibility Workbench. In your browser, access the add-ons menu (usually under the Tools menu) and enable the Sterling Designer extension. A popup dialog box opens, asking you to install two add-ons (jsLib and the Sterling Designer extension). Make sure that you install both of the add-ons, and then enable them.

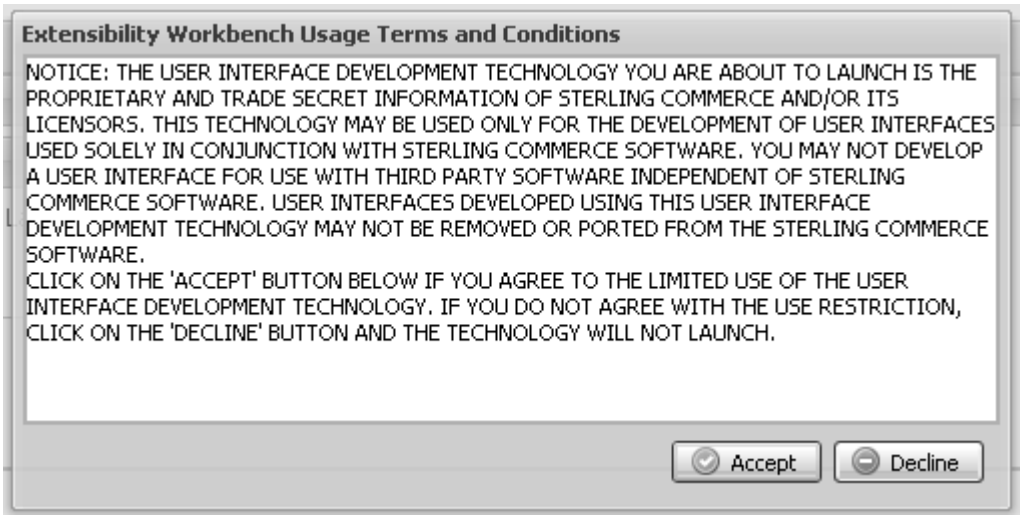
This option to install the add-ons is only available in Mozilla Firefox. Internet Explorer, the other browser supported by the Extensibility Workbench, uses ActiveX for reading and writing files. In IE, make sure that the ActiveX settings are correctly enabled.

When logging in to the application console from IE, if you get the *Could not use ActiveX for file IO* warning, then the IE settings for ActiveX must be checked to ensure that all relevant settings are enabled. If these settings are not enabled, you will not be able to view any folder displayed in the Mapping Preferences dialog box while setting the workspace directory for the Extensibility Workbench.

Make sure that the server/site which is hosting the application is added under the secured/trusted sites list in IE.

Using the Web UI Framework Extensibility Workbench to Modify a Widget

1. In the application, open the screen that you want to change.
2. Click **Shift + space bar**.
3. Review and accept the following terms and conditions:

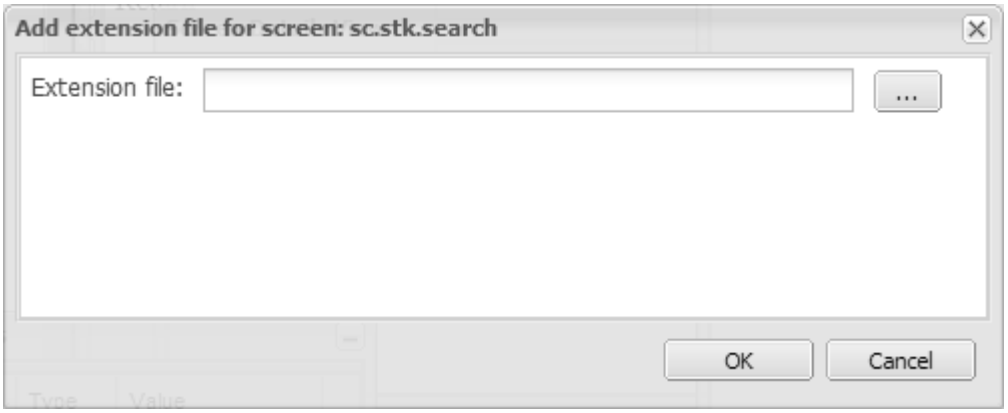


The Extensibility Workbench launches if you accept the terms and conditions (provided you have also completed the Directory to URL Mapping and you have associated a file for the screen). Any subsequent use of the **Shift + space bar** hot key hides and then re-launches the workbench until the browser is refreshed or a new screen is opened, in which case the Terms and Conditions window re-appears.

If you decline the terms and conditions, the workbench does not launch.

The tools of the Extensibility Workbench appear in different views.You might have to click the plus sign button on the Extensibility Workbench toolbar to display all of the views.

4. Click the plus sign button to show all views or the minus button to minimize all views. When all of the views are minimized, you can click the tab of a view to display just that view. When a view is displayed, you can minimize the view by clicking the minus sign in the upper right hand corner of the view.
5. Before you can work with a widget on a screen, you need to select or associate the extension file for the screen. The extension file stores the extensions (changes) to the screen. The **Add extension file for screen** dialog box appears when you first try to work on a widget.



The extension file contains metadata about your changes. Extension files are saved in your current working directory. They can be viewed in the Files tab of the Palette & Files view.

When the Extensibility Workbench is launched for the first time, the current working directory is defaulted to the directory entered during mapping. You can later change this in the Files tab.

a) In the **Add extension file for screen** dialog box, specify the extension file by either using the browse button to select an existing file, or by typing the name of the file in the **Extension file** field.

   Type the name of the file if you want to associate/create a new file. If you have already extended the screen and have an extension file for the screen, you can browse for the file.

b) Click the **OK** button.

   If a dialog box appears that includes the message *Selected file contains source that does not match with the current screen.*, click **OK** to overwrite the file or **Cancel** to choose another file. This message usually means that you have chosen the wrong metadata file.

After you have saved the extensions to a screen and deployed those changes in the application, you do not have to add the extension file to make further changes to the screen. The extension file will be automatically loaded with the screen.

After you select this extension file, the following view actions occur:

- The Outline view populates with information about the widgets on the screen.
- The Screen Details View populates with information about the extension file.

  The Extension Class Name field displays the name of the generated extension class. You can change this name.

- The Properties view displays the original properties of the widget.

6. To add a widget to the screen, select the widget on the Palette tab. Right-click or left-click at the place on the screen where you want the widget to appear.

   When you add a new widget, the sciId property of the widget must include the default **extn_** prefix. This differentiates an extended component from an out-of-the-box component.

7. To change a widget, select it on the screen or in the Outline view.

8. To change or create a widget property, do the following. For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.
   - If you want to change the properties of an item, click the **Refresh instances** button in the Screen Details view to make those property changes active.

     If you want to change the properties of an existing item, a separate grid titled “Original Properties” appears that displays the original property values of that item.

     The “Original Properties” grid is not shown for any new component added through extensibility. Any new properties added through extensibility are listed in a separate grid above the "Original Properties" grid (if the component is present in the base screen).

     You cannot change a property listed under “Original Properties”. However, you can override an existing property or add new properties to an out-of-the-box component.
   - To create a new property for an item, click the **Add** button in the Properties view. Before clicking the **Add** button, the desired property should be selected from the dropdown list of available properties.
9. To save your changes in your project directory (but not deploy them), click the **Save** button in the Screen Details view.
10. To work on another screen, go to the other screen and then re-activate the Extensibility Workbench by clicking **Shift + space bar**.
11. After you have saved all of your screen extensions, you must deploy the changes for them to take effect in the application.

Extensibility Workbench Tools of the Web UI Framework

The following table shows the tools to use for the different tasks that you can perform using the Extensibility Workbench:

For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

If you want to...	Use this tool...
Add widgets to a screen.	Palette & Files view (Palette tab)
Create input and output data sources for mashup layer files.	Palette & Files view (Data tab)
Access the local file system.	Palette & Files view (Files tab)
View a directory-type listing of all the widgets on the screen. The view populates after you specify an extension file and select or add an item.	Outline View (Components tab)
Re-arrange UI components in a directory-type listing.	Outline View (Components tab)
Show overlays applicable for the screen.	Outline View (Overlays tab)

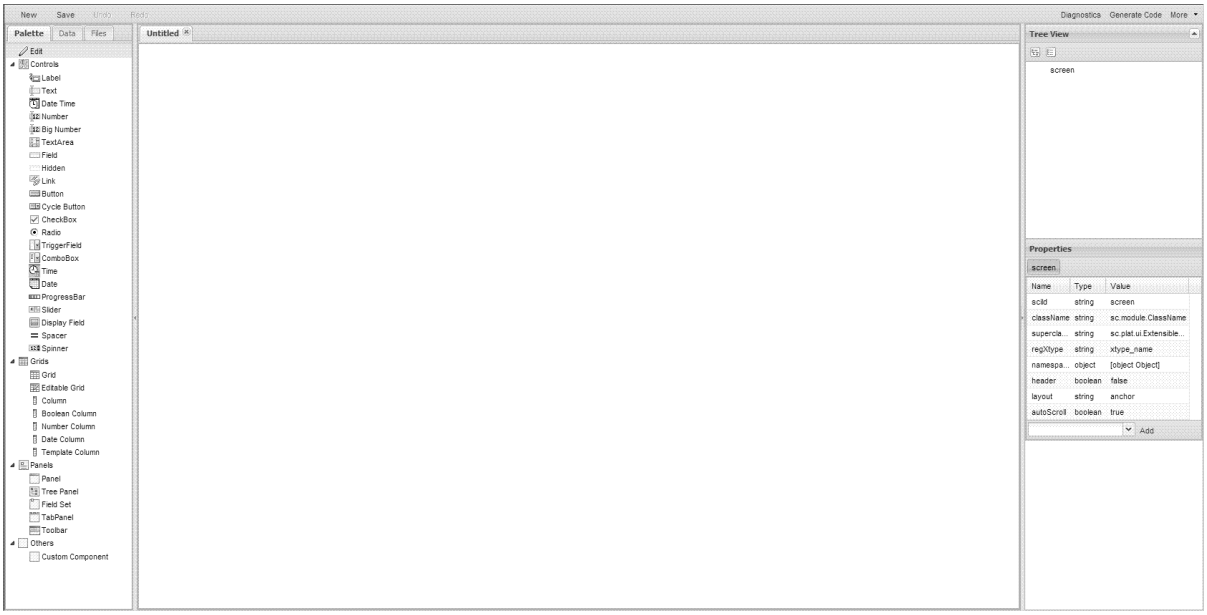
If you want to...	Use this tool...
This shows all of the changes made with the workbench. It helps you view changes and (if necessary) remove them. The **Click to View Overlays** button displays overlays.	
Collapse the Outline view to show just the top screen item (screen).	
Expand the Outline view to show all of the screen items.	
View a list of the properties of the screen and any widget that is selected.	Properties view
Add a property to a widget.	Properties view (Add tab)
Show the name of the screen that is being modified.	Screen Details view
Save an updated screen.	Screen Details view (Save button)
Activate changes to the properties of a widget.	Screen Details view (Refresh instances button)
Access bundle files for localization.	Screen Details view (Localize button)
Start or stop extensibility. You can change screens (by selecting the appropriate menu option) without having to use this button.	
Display all views.	
Minimize all views.	
Link to the Designer Workbench.	**Design new screens** link

Designer Workbench of the Web UI Framework for Custom Developers

The Designer Workbench allows you to use WYSIWYG tools to build new screens for the application. It has tools similar to the Extensibility Workbench, which is used to change the screens of an out-of-the-box installation of the application.

Custom developers access the Designer Workbench by clicking the **Design new screens** link in the Extensibility Workbench.

Note: Although you can access the Designer Workbench from an out-of-the-box installation of the application, you are limited in the changes that you can make. Please contact Sterling Commerce Customer Support when changing an out-of-the-box installation using the Designer Workbench.



Creating New UI Screens Using the Designer Workbench in the Web UI Framework

Use the canvas in the Designer Workbench to create the actual user interface that will be used by an application. Work with the canvas by dropping (adding) widgets from the Palette view of the workbench. You cannot use the Extensibility Workbench to create new screens.

Use the buttons in the upper left hand corner of the Designer Workbench to do the following:

- New

  Create a new screen.

- Save

  Save the changes on a screen.

- Undo

  Undo screen changes that you have not saved yet.

- Redo

  Redo changes that you have undone using the **Undo** button.

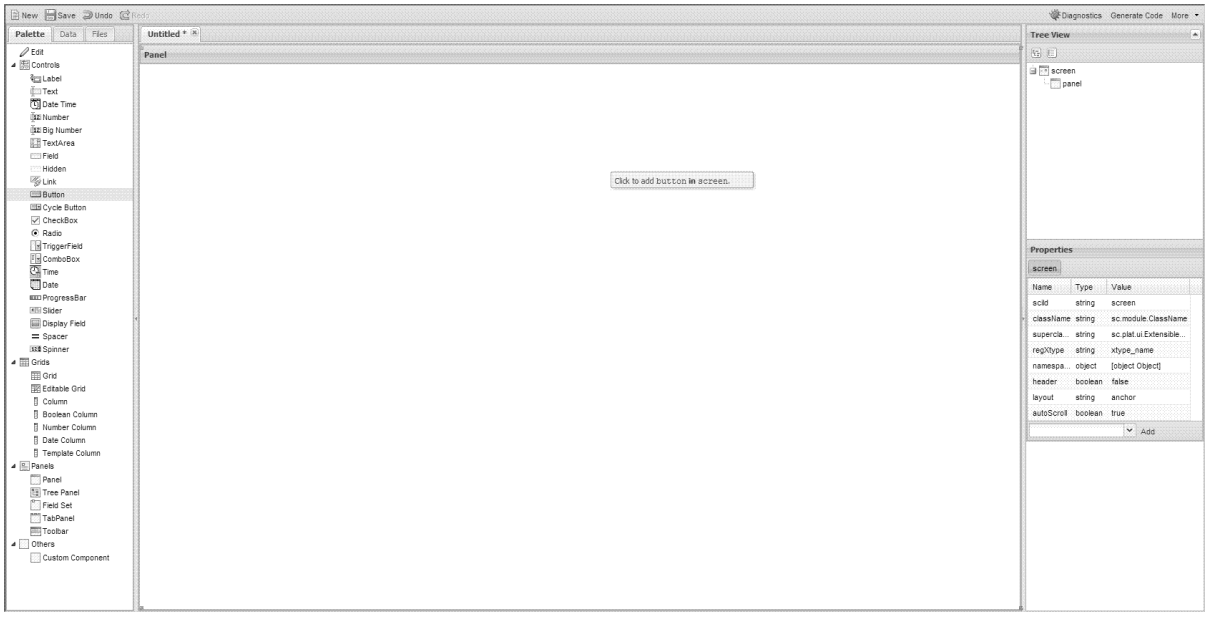
Follow these guidelines when dropping widgets:

- Make sure that the widget is selected in the Palette view before you drop it.
- Make sure your cursor is over the canvas before you drop the widget on the canvas.
- Use the canvas tooltips to decide when to drop and how to drop the widget. You cannot drag and drop a widget from the Palette view to the canvas.

  For example, if your screen includes a panel, and you want to add a button to screen (but not to the panel), make sure:

  - Your cursor is not over the panel.

- The tooltip reads **Click to add button in screen** and not **Click to add button in panel** or **Click to add button before panel**.



- Use the Tree View to delete or re-arrange the widgets. To delete a widget, you must first right-click the widget and select the delete option.
- Use the widget names in the Palette view to create preset properties, but right-click the widget in the canvas to apply a preset property.

Designer Workbench Tools of the Web UI Framework

The following table shows the tools to use for the different tasks that you can perform using the Designer Workbench:

For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

If you want to...	Use this tool...
Create a new screen.	New button
Save a new screen.	Save button
Undo changes that you made to the screen.	Undo button
Redo changes to the canvas that you just undid.	Redo button
Add widgets to a screen.	Palette tab
Create data sources for sending data to input and output XML files. Mashup layer files use these data sources.	Data tab
Add file access to a control.	Files tab
Configure the directory path to the project that will use the changes from the Designer Workbench.	Files tab (Options button)

If you want to...	Use this tool...
Immediately update your main project with changes that you make using the Designer Workbench.	Files tab (Notify project checkbox)
View the workspace for the screen that you are creating by adding widgets.	Canvas
View a directory-type listing of all the widgets on the canvas (screen).	Tree View
Collapse the Tree View to show just the top screen item (screen).	
Expand the Tree View to show all of the screen items.	
View a list the properties of the screen and any widget that is selected.	Properties view
Add a property to a widget.	Properties view (Add tab)
Check if a screen has any errors or warnings and see if a fix is available, using the Check results dialog box. For example, if you have not localized all of the controls in the current screen, the Check results dialog box displays a list of controls that have not been localized. To resolve this, click the icon under the Fix? column, which directs you to the Localization Panel, where you can localize the controls.	Diagnostics button
Create mashup layer files. This button displays the Configure Mashups dialog box.	Mashups button
Display JavaScript source code for the screen.	More button (View Source menu option)
View an encoded string of user preferences that are stored as cookies (like the project directory and data source directory). This string can be copied and added to your JavaScript bookmarks. If you clear all of your browser cookies, you can use this user preference information to restore your original preferences.	More button (Export Preferences menu option)
Localize widgets. This button displays the Localization panel dialog box. You must first save the screen before you can localize any widgets.	More button (Localize Screen menu option)
Load libraries into the Designer Workbench. To do this, you must add an include file.	More button (Manage Libraries menu option)
Generate code from the Code Template Generator window that you can use to update mashup, Struts, JSB, resource, resource permission, and menu files.	Generate Code button

Using the Web UI Framework Designer Workbench from the Extensibility Workbench to Create New Screens for Custom Developers

1. Access the Designer Workbench from the Extensibility Workbench by clicking the **Design new screens** link in the lower right hand corner of the Extensibility Workbench.

2. In the Designer Workbench, click the **New** button to create a new screen.
3. Perform one or more of the following tasks:
   - To add a widget, click on the Palette tab. Select a widget. On the canvas, right-click or left-click where you want the widget to appear. You can later use the Tree View to rearrange the order of the widgets.
   - Use the Tree View to see a directory-style overview of how widgets are arranged on the canvas. Also use the Tree View to delete items or rearrange items (for example, move a column from one grid to another grid, or move a button from one panel to another panel).
   - Use the Properties view to add or change any widget properties. For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.
   - (Optional) To create a mashup, click the Data tab. Configure an output data source. Then, click the **Mashups** button to create the mashup xml file, which will include a reference to the output data source.
   - (Optional) Use the Code Template Generator window to generate code that can be either pasted on the Code Update page and updated in the application or (if the changes are permanent) saved in a relevant file like mashup.xml or a Struts file for the application. Access the Code Template Generator window using the **Generate Code** button near the **More** button.
4. To specify the project that will use the screens that you are creating and modifying, click the Files tab.
   a) Check the **Notify project** checkbox.
   b) Click the **Options** button.

The Configure Project Directory dialog box appears.

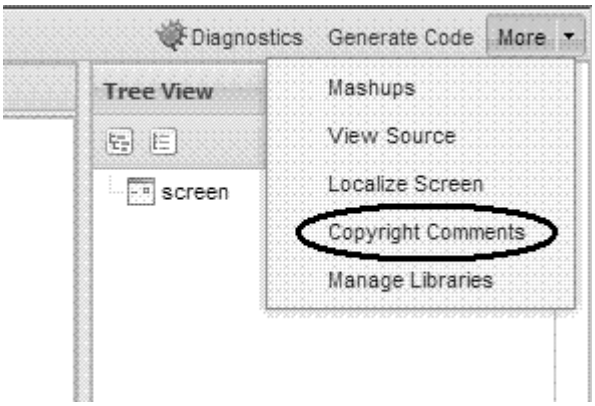
c) Configure your project directory and click **OK**.

5. To save the new screen, click the **Save** button. The changes immediately appear in the project file.
6. To return to the Extensibility Workbench, use the **Back** button of your browser. You will have to re-activate the Extensibility Workbench by clicking **Shift + space bar**.
7. After you create all of your new screens, you must deploy them as an extension to the application. For more information, refer to the documentation on deployment.

Generating Copyright Comments with the Web UI Framework

Copyright comments can be added to all of the js and config.js files which get generated through the Designer Workbench.

1. Click the **More** button at the top right of the Designer Workbench toolbar.



2. Click **Copyright Comments**.

   This brings up a window to provide the copyright comments.

3. Enter comments. For example, **Copyright Sterling Commerce 2010**.
4. Click the **Save** button to save the comments.

   The comments are stored in the designer.preferences file at: C:\Documents and Settings\<*user-name*>\.designer. If you delete the designer.preferences file, you need to provide comments again, because this file holds all of your Designer Workbench-related preferences.

   Sample comments generated:

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* Copyright Sterling Commerce 2010
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

   where

```
Copyright Sterling Commerce 2010
```

   are the comments provided by the user.

   Once this setup is done, all files (both old and new) that are saved through the Designer Workbench are generated with these comments.

   These copyright comments are also used and generated in the overlay files generated through the Extensibility Workbench and the bundle files during localization.

Note: It is recommended that you delete browser cache files and cookies before entering copyright comments.

Note: If you remove the provided comments in the **Provide Copyright Comments** dialog box and click **Save**, or if you don't do this setup, your screens will be generated the way they were being generated before, that is, without any copyright comments.

Mashup Layer of the Web UI Framework

The mashup layer of the Web UI Framework connects end user actions at the front end of an application with business logic at the back end of an application. The mashup layer handles authorization (permission control) and transaction management tasks. You can configure the mashup layer in the Designer Workbench. The mashup layer allows business calls for data handling to be used in more than one place (like a backend server and a database) without being repeated.

The mashup layer is a core logic service layer that acts as an intermediate service layer to which both action classes and JSON (JavaScript Object Notation) endpoints delegate. Examples include the Struts action classes in the UI backend and the Ext JS JSON in the presentation layer. The action classes and the JSON endpoints act as types of adapters, with the core logic contained in the service layer.

The mashup layer of the Web UI Framework is used to invoke business calls (XAPI calls) for data handling to the backend server. Each XAPI call and multiple XAPI call are always called under one transaction. Each multiple XAPI is under one transaction. The mashup layer is a mixture of XAPIs and other mashups.

Sterling Commerce recommends that you use the mashup layer of the Web UI Framework. If you do not use the Web UI Framework, a mashup layer exists, but it does not support XAPI calls to the backend server.

The mashup layer does not contain business logic. Its main purpose is to call different APIs and create data that is user interface-specific. You can access the mashup layer from the user interface back end with different development tools (Struts, DWR, custom servlet, etc.).

The mashup layer can do the following:

- Handle calls to the business logic layer to get or modify data.
- Take responsibility for bean creation and then invoke the business logic layer.
- Take responsibility for managing data transformation so that the output data is ready for use in the presentation layer.

If errors occur in the following situations, check your mashup setup:

- If the mashup metadata is not found for a given mashup ID.
- If mashup metadata is not extensible, but an attempt is made to extend it.

Interface Contracts of the Web UI Framework - Mashup Layer

For more information, refer to the Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

Interface Contract	Description	Methods
ISCUIMashup	Includes the business logic of an application. If transactional is set, then all of the business logic in one mashup will be under one transaction. A custom mashup implementation is plugged in using an <app>_mashup.xml file. Any XAPI service calls that an application might need for its business logic will be included in the <API> element of the <app>_mashup.xml file.	• execute Takes in SCUIContext, input object, and XML in the form of metadata as an SCUIMetaData object.

Mashup Layer Classes of the Web UI Framework

Class	Description	Methods
SCUIMashupRegistry	Helps load mashup during initial setup. Reads all of the mashup XML files and creates SCUIMashupMetaData objects per mashup ID. It maintains this registry for all of the mashup IDs according to whether a mashup is extensible.	• loadMashup Loads all the XML files under the /mashupxmls/<applicationId> directory in the context root. • loadExtnMashup Loads all the XML files under the /mashupxmls/<applicationId>/extn directory in the context root.
SCUIMashupHelper	Called by a Struts action to load / fetch mashups.	• invokeMashup(String mashupId, SCUIContext uiContext Object input) If resourceId is given, calls the authorization layer.

Class	Description	Methods
		If transactional is set to true, sets the transaction context. Instantiates the mashup implementation given by the class name in the mashup.xml file. • loadMashupXML Called by the startup servlet.

Mashup XML Metadata of the Web UI Framework

A mashup configuration is an XML file that you create in the Designer Workbench. This XML file includes the following items:

XML Item	Type of Item	Description
mashups	Element	Encloses all of the details of a mashup. Contains the definition of one or more individual mashups in an element mashup.
id	Attribute (mashup tag)	Unique identifier of the mashup.
transactional	Attribute (mashup tag)	Indicates if the mashup is transactional in nature (true if transactional). Used for the transaction management task. For all out XAPI calls, this must be set to **true**.
description	Attribute (mashup tag)	Describes the mashup.
resourceId	Attribute (mashup tag)	Unique identifier of the resource which needs to be authorized. Used for the authorization task. If resourceId is not specified, authorization does not take place and the mashup gets the permission by default. Create resources in the Application Manager. If a mashup is given access to all resources, the resourceId is not needed. If a resourceId does not have permissions for a mashup, it cannot view the results of that mashup. This will result in the message `Mashup invocation failed.` If two mashups have the same ID and namespace names, the mashup is invoked only once. If two mashups have the same ID, but different namespace names, the mashup is invoked two times. If the permission for one tag of a mashup is revoked, the mashup cannot be invoked.
extensible	Attribute (mashup tag)	Indicates if the mashup can be extended.
mashuptype	Attribute (mashup tag)	Has the following values: • XAPI (for XAPI calls) • AggregateXAPI (for multiple mashups)

XML Item	Type of Item	Description
classInformation	Tag (within mashup tag)	Includes a name attribute, which is the fully qualified class name of the mashup implementation.
mashupRef	Tag (within mashup tag)	Indicates each XAPI within a multiple XAPI call in a mashup. Includes an id attribute and a namespace tag (APINamespace).
APINamespace	Tag (within mashupRef tag)	Defines namespace for each XAPI in a multiple XAPI call. Includes the following attributes: • inputNS - input namespace • outputNS - output namespace

The following are examples of mashup XML files:

```
<mashups>
     <mashup id ='m0001'
          transactional='true'
          resourceId='SC02187'
          extensible='true'
          mashuptype='XAPI'>
      <classInformation
name="com.sterlingcommerce.ui.web.framework.mashup.impl.SCUIMashupImplementer"
/>
      </mashup>
      <mashup ...
      />
      <mashup ...
      />
</mashups>
```

```
<mashups>
      <mashup id='STK-getAllFlightInfo' transactional='true'
       description="Flight, Flight Trips, Flight servicesmashup"
       mashuptype='AggregateXAPI'>
         <classInformation
name="com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIAggregatorMashup" />
         <mashupRef id="demoapp-stk-getFlightList">
             <APINamespace inputNS='flight'
             outputNS='flightOutput' />
          </mashupRef>
          <mashupRef id="demoapp-stk-getFlightServiceList">
              <APINamespace inputNS='flightService'
              outputNS='flightServiceOutput' />
           </mashupRef>
           <mashupRef id="STK-aggFlightTrip">
               <APINamespace inputNS='flightTrip'
                outputNS='flightTripOutput' />
           </mashupRef>
        </mashup>
</mashups>
```

Configuring Mashups in Web UI Framework

1. Open the Designer Workbench.
2. Click the Data tab.
3. Configure an output data source.
4. Click the Mashup button to create the mashup XML file, which will include a reference to the output data source.

Specifying Multiple XAPI Calls with the Web UI Framework

With the Web UI Framework, you can specify more than one XAPI call under one transaction, using the mashup layer. When you do this, you create mashups within other mashups. Sterling Commerce recommends that you use multiple XAPI mashup configurations only for fetch operations, and not for save operations.

1. Open the Designer Workbench.
2. Click the **Mashups** button to create or open a mashup.xml file.
3. For each XAPI, create mashups within the main mashup. Each of these mashups has an id as an attribute to uniquely identify the mashup definition in the XML file. Each mashup also has a resourceId as an optional attribute which is used for authorization and takes precedence over individual resource permission defined under the main mashup element.

   Also, these mashups contain one or more mashupRef elements, which are used to reference other mashups. The mashup id referenced in the mashupRef element must be defined in the mashup.xml file before the reference.

   The mashupRef element can also have an endpoint as an attribute that will take precedence over endpoint attribute in the mashup element. The endpoint attribute in the mashup element will in turn take precedence over the one defined in the XAPI layer.
4. In each mashupRef tag within each XAPI mashup, use the APINamespace element to define the input namespace and the output namespace for each API in the mashup. If this element is not given, the input namespace defaults to the element name that serves as the input XML and the output namespace defaults to the element name in the output XML.

   If there are two mashups with the same id values and same namespaces, the calls are merged to only one XAPI under the multiple XAPI call. If the namespaces are different, they are treated as separate XAPIs under the multiple XAPI call.

Example of mashup.xml File with Multiple XAPI Calls in the Web UI Framework

```
<mashups>
    <mashup id ='m0001'
        resourceId='SC02187'  extensible='true'
        endpoint='myHttpEndpoint'
        mashupType='XAPI'>
        <classInformation
 name="com.sterlingcommerce.ui.web.framework.mashup.impl.SCUIXAPIMashup"/>
        <API Name="getFlightList">
        <input>
```

```
        ...
        </input>
        <template>
        ...
        </template>
        </API>
        <APINamespace inputNS='inputServiceKeys'
        outPutNS='outputServiceList' />
   </mashup>
   <mashup id= 'm0002'...
   />
   <mashup id= 'm0003'...
   />
   <mashup id='mm001'  resourceId='SC05457' mashupType='AggregateXAPI'>
       <mashupRef id = 'mm001'>
                <APINamespace inputNS='inputServiceKeys'
                outPutNS='outputServiceList' />
        </mashupRef>
        <mashupRef id = 'm0002'/>
    </mashup>
    <mashup id='mm002' mashupType='AggregateXAPI'>
       <mashupRef id = 'mm001'>
           <APINamespace inputNS='inputServiceId'
           outPutNS='outputServiceListForId' />
        </mashupRef>
        <mashupRef id = 'mm001' />
    </mashup>
</mashups>
```

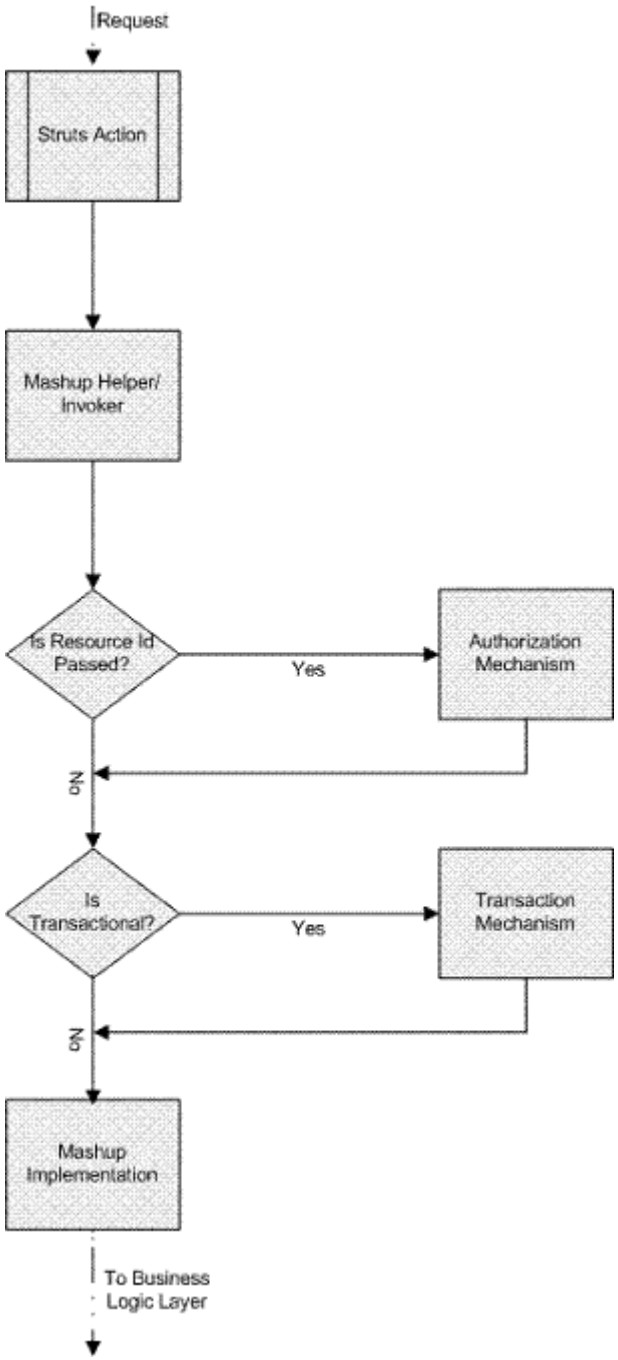
How the Mashup Layer Handles Authorization and Transaction Management in the Web UI Framework

The mashup layer acts as a single point where authorization and transaction management are handled in a consistent way, no matter what development tool you use to access it (Struts, DWR, custom servlet, etc.).

If a resource ID is not passed for authorization or if the request is not transactional, the request is not stopped. Instead, it continues through to the mashup and business logic layers.

Each invocation of a mashup is considered to be part of a transaction.

The following graphic shows how the mashup layer handles authorization and transaction management, using the Struts development tool as an example:



Extending Mashups in the Web UI Framework

You can extend mashups using both differential and override extensibility. A mashup is extended on the basis of the <mashup id> tag specified in the XML file. A XAPI mashup can be extended irrespective of the screen being extended.

Extending Mashups Using Override Extensibility in the Web UI Framework

You can extend a mashup using override extensibility both automatically and manually.

In both of the following procedures, you create a new mashup XML file to replace the mashup file that you are extending. You can create that file in one of the following ways:

- Hand-coding.
- Generation of code using the mashup template in the Code Template Generator. The mashup id would be the same as the id of the mashup file that you are extending. After you paste the generated code on the Code Update page, you can test the behavior of the new mashup file in the application.

1. To automatically override a mashup, do the following:
   a) Identify a mashup to be extended.
   b) Go to the <*INSTALL_DIR*>/extensions/<*app_dir*>/webpages directory. This directory path is not part of the out-of-the-box installation, and must be created by the user within the <*INSTALL_DIR*>/extensions directory.
   c) Replicate the relative folder structure (relative with regard to deployment) of the mashup XML file containing the mashups to be created. The original mashup XML file is located in the <*INSTALL_DIR*>/repository/eardata/<*app_dir*>/war/mashupxmls/<*app_dir*> directory.
   d) Create a new XML file with the same name as the base file. Any mashup in this XML file that has the same ID as the base file would override the base file mashup.

2. To manually override a mashup, do the following:
   a) Create a new mashup XML file with entries for the mashup file to be overridden. This file can have any name and does not need to replicate the relative directory structure of the XML file containing the mashup to be extended.
   b) Add this XML file to the <*INSTALL_DIR*>/extensions/<*app_dir*>/webpages directory.

      If you have created a servlet class to register a JSB (JavaScript Builder), the code to include this mashup XML file also can be written in the same servlet class. Else, you should create a servlet class to register this mashup XML file. Use the method loadOverrideMashupXml in the SCUIMashupHelper class. For more information, refer to the documentation on deploying extensions using JavaScript Builder files.

Extending Mashups Using Differential Extensibility in the Web UI Framework

To extend a mashup using differential extensibility, do the following:

1. Create a new mashup XML file with entries for the mashup file to be overridden. This file can have any name and does not need to replicate the relative directory structure of the XML file containing the mashup to be extended.
2. Add this XML file to the <*INSTALL_DIR*>/extensions/<*app_dir*>/webpages directory. This directory path is not part of the out-of-the-box installation, and must be created by the user within the <*INSTALL_DIR*>/extensions directory. The original mashup XML file is located in the <*INSTALL_DIR*>/repository/eardata/<*app_dir*>/war/mashupxmls/<*app_dir*> directory.

   If you have created a servlet class to register a JSB (JavaScript Builder), the code to include this mashup XML file also can be written in the same servlet class. Else, you should create a servlet class to register

this mashup XML file. Use the method loadIncrementalMashupXml in the SCUIMashupHelper class. For more information, refer to the documentation on deploying extensions using JavaScript Builder files.

The new file's contents are added to the respective mashups in the base screen based on the <mashup id>.

Creating and Extending a Struts XML File in the Web UI Framework

1. Create your app_extn_struts.xml file to extend the app_struts.xml file which contains all of your actions.
2. Navigate to the <Install Dir>/repository/eardata/<application name>/extn directory and re-name the struts.properties.sample and struts.xml.sample files to **struts.properties** and **struts.xml** respectively.

   The following shows struts.properties sample contents:

   ```
   struts.action.extension=do

   struts.devMode=true
   ```

   The following shows struts.xml sample contents:

   ```
    <struts>

          <include file="struts-default.xml"/>

          <include file="scuiimpl_struts.xml"/>

          <include file="app_struts.xml"/>

          <include file="app_extn_struts.xml"/> <!--your extn struts must be
   included after the app_struts.xml -->

   </struts>
   ```

3. Include the app_extn_struts.xml file in the classpath. This can be done by one of the following ways:
   - Create a WEB-INF/lib directory in the extn directory and copy your jar file containing the app_extn_struts.xml file there. This step can be followed in case of single and multiwar deployments.
   - Create a WEB-INF/classes directory in the extn directory and copy your app_extn_struts.xml file there. This step can be followed in case of single and multiwar deployments.
   - Create a jar file containing the app_extn_struts.xml file and run the Install3rdParty.sh script. This step can only be followed in case of a single war deployment.
4. Run the buildear or buildwar utility to create the EAR/WAR file.

Creating a Menu Entry for a New Web UI Framework Screen Using the Application Manager

You can use the Application Manager to create a new menu entry for a new Web UI Framework screen using the following items:

- Resource

• Menu
• User permissions
• Struts xml file

You can also create a new menu using the Code Template Generator of the Designer Workbench. Use the Code Template Generator to access the Code Update page, where you create the menu using code that you generated in the Code Template Generator. For more information, refer to the documentation on the Code Template Generator.

1. Launch the application.
2. Launch the Application Manager.
3. In the Application Manager, click **Applications** > **Platform**.
4. Create a new resource by doing the following:
   a) Double-click the **Presentation** item.
   b) Double-click the **Resources** item.

      The Resource Hierarchy appears.

   c) Select the **Sterling_Supply_Chain_Applications_Console** item.
   d) Click the **Create New** button (the green plus sign).

      The Resource Details screen appears.

   e) Type information for all the tags.

      The Resource ID tag associates menus and resources. For the URL tag, type **<package namespace in your struts.xml file>/<action name>**. For the Resource Type tag, select **StrutsAction** from the dropdown list.

      Note: The URL package name and the action name in the struts.xml file should be the same.

   f) Click the **Save** button in the upper right corner of the Resource Details screen.
5. Create the new screen using the Designer Workbench.
6. Copy all of the generated files of the new screen to a new folder in the <*app_dir*>/webpages directory. These files include the <*newscreen*>.json, <*newscreen*>.js, <*newscreen*>_config.js, and <*newscreen*>.js.sample files.
7. Create a new menu by doing the following:
   a) Double-click the **Presentation** item.
   b) Double-click the **Menu** item.

      The Menu Hierarchy appears.

   c) Double-click the option for the menu where the new screen will be accessed.

      For example, you would double-click the **<*application*>_Admin_Menu** option to create a menu under the top menu or under an existing submenu like AdminPage.

   d) Click the parent menu for the new menu entry.
   e) Click the **Create New Menu Item** button (it includes a green plus sign).

      The Menu Item Details screen appears.

   f) Type information for all the tags.

      For the Resource ID tag, select the resource with which this menu should be associated.

8. Give user permissions by doing the following:
   a) Double-click the **Security** item.
   b) Double-click the **Users** item.

      The User Search screen appears.

   c) Select a user and subscribe to a group. For example, you could select **<*application*>admin** and subscribe to the SYSTEM group.
   d) Under the Security item, double-click the **Groups** item.

      The Groups screen appears.

   e) Edit the details for the user's default group. For <*application*>admin, the default group is <*application*>admingroup.
   f) Double-click the default group name to display the Group Details screen.
   g) Click the **Permissions** button for the Cross Application option.
   h) Allow the user access to the new Struts action.
   i) Save the changes and revert the group subscriptions to the default values.
9. Define the Struts action in a Struts config file which serves the page that is linked to where you click on the menu. You can use the Struts file in the Code Template Generator to define the Struts action, which you would then paste into the above mentioned config file. The resourceId should be the same as the resourceId defined in the Application Manager.

   For more information on how to include this file entry in the struts.xml file, refer to the information on creating and extending Struts XML files.

   The jar file for the install3rdParty.sh command should also contain the java class file for this Struts action.

```
<struts>
    <package name="<package-name>" namespace=/<namepsace>""
extends="struts-default">
          <action name="home" class="<struts-action-class>">
                    <param name="RessourceId"><resourceId></param>
                    <result name="success"><result-1></result>
          </action>
    </package>
</struts>
```

Deploying Web UI Framework Extensions

After you customize an existing screen using the Extensibility Workbench or create a new screen using the Designer Workbench, you must deploy your changes in the application. You can use either Java Server pages (JSP) or JavaScript Builder (JSB) files to deploy Extensibility Workbench changes. To deploy Designer Workbench changes, you must use a Java Server page.

A JavaScript Builder file contains JavaScript library/package definitions. The Web UI Framework provides programmatic control over this library with differential extensibility.

If minification is required, the directory structure in the <Install dir>/extensions/<app name>/webpages directory changes slightly. For more information, refer to the documentation on compiling and minifying JavaScript files.

Deploying Extensions Created by the Web UI Framework Extensibility Workbench and Designer Workbench Using a Java Server Page

Do the following to use a Java Server page (JSP) to deploy differential extensions (modified with the Extensibility Workbench) or override extensions (created using the Designer Workbench):

Note: UNIX/Linux file paths are used in the following procedure.

1. Install the application and build a WAR file for it.
   a) Deploy the WAR file on the server in the exploded format.
   b) After the deployment finishes, start the application server.
2. Make sure that the changes made using the Extensibility Workbench or the new screen created using the Designer Workbench have all the relevant JSON and JavaScript files generated and saved.
3. In the <*INSTALL_DIR*>/extensions folder, create the following subdirectory:

   <*application package name*>/webpages
4. In the webpages subdirectory, replicate the directory structure of the screen that you want to extend (relative to your deployment) and copy in all of the script files generated by the workbench.

   For example, if you extend the Manage Flight Route screen (which uses the file path <*application package name*>/flightRoute), you would copy all of the extension script files into the <*INSTALL_DIR*>/extensions/<*application package name*>/webpages/<application package name>/flightRoute directory.
5. Create a new JSP file with the same name as the base JSP file to launch these newly generated files in the same folder.

   Sample code for the original JSP:

```
<%@ taglib uri="/WEB-INF/scui.tld" prefix="scuitag" %>
<jsp:include page="/stk/include.jsp">
    <jsp:param name="title" value="manage flight route" />
</jsp:include>
<script>
    Ext.ns("sc.stk");
    sc.stk.fn = function() {
        var fr = new sc.stk.flightRoute();
       sc.plat.ScreenTitle.setText(fr.Header, null, "sc-panel-belowmenu-text");

       sc.plat.ScreenTitle.setText(fr.Header, null, "sc-panel-belowmenu-desc");

        fr.render("mainBodyPanel");
    }
<scuitag:includeJS
name="['/stk/flightRoute/flightRouteList_config.js','/stk/flightRoute/flightRouteList.js',
'/stk/flightRoute/flightRouteList_bundle.js']"
callBack="sc.stk.fn"/>
</script>
<jsp:include page="/stk/footer.jsp">
```

Sample code for the new JSP (differential extensibility):

```
<%@ taglib uri="/WEB-INF/scui.tld" prefix="scuitag" %>
<jsp:include page="/stk/include.jsp">
    <jsp:param name="title" value="manage flight route" />
</jsp:include>
<script>
    Ext.ns("sc.stk");
    sc.stk.fn = function() {
        var fr = new sc.stk.flightRoute();
       sc.plat.ScreenTitle.setText(fr.Header, null, "sc-panel-belowmenu-text");

        sc.plat.ScreenTitle.setDescription(fr.Header, null,
"sc-panel-belowmenu-desc");
        fr.render("mainBodyPanel");
    }
<scuitag:includeJS
name="['/stk/flightRoute/flightRouteList_config.js','/stk/flightRoute/flightRouteList.js',
'/stk/flightRoute/flightRouteList_bundle.js','/stk/flightRoute/test_overlays.js'
,'/stk/flightRoute/test.js']"
callBack="sc.stk.fn"/>
//The new JSP also includes the newly generated files: test_overlays.js and
test.js
</script>
<jsp:include page="/stk/footer.jsp">
```

Sample code for the new JSP (override extensibility):

```
<%@ taglib uri="/WEB-INF/scui.tld" prefix="scuitag" %>
<jsp:include page="/stk/include.jsp">
    <jsp:param name="title" value="Custom Screen" />
</jsp:include>
<script>
    Ext.ns("sc.stk");
    sc.stk.fn = function() {
        var fr = new sc.stk.flightRoute();
/*
sc.extn.CustomScreen is the className for the new screen.  It is available
as a property for a screen in the designer and defaulted to
sc.module.ClassName.  The user can change it.
*/
       sc.plat.ScreenTitle.setText(fr.Header, null, "sc-panel-belowmenu-text");
/*
Here, the setText(arg1, arg2, arg3) method has been used to set arg1 as
title for a page as arg3="sc-panel-belowmenu-text".  Here, a bundle entry in
the file: newScreen_bundle.js corresponding to Header would be picked up.
*/
       sc.plat.ScreenTitle.setDescription(" ", null, "sc-panel-belowmenu-desc");

        fr.render("mainBodyPanel");
    };
<scuitag:includeJS
name="['/extn/stk/flightRoute/newScreen_config.js',
'/extn/stk/flightRoute/newScreen.js', '/stk/flightRoute/newScreen_bundle.js']"
callBack="sc.stk.fn"/>
```

```
//This JSP includes the files generated through the designer:
newScreen_config.js, newScreen.js and the localization file:
newScreen_bundle.js
</script>
<jsp:include page="/stk/footer.jsp">
```

6. Rebuild the WAR file.

   The contents of the <*INSTALL_DIR*>/extensions/<application package name>/webpages directory are copied to the following directory:

   <*INSTALL_DIR*>/external_deployments/<application package name>/extn

   This directory structure exists only if a WAR file is created and then exploded in the same <*INSTALL_DIR*>/external_deployments directory.

   Any JSP file within this directory that has the same name and at the same relative directory structure as the base JSP would override the out-of-the-box JSP file.

7. Relaunch the application to display the extended changes.

Deploying Extensions Created by the Web UI Framework Extensibility Workbench Using a JavaScript Builder File

Do the following to use a JavaScript Builder file to deploy differential extensions created using the Extensibility Workbench. You cannot use this procedure to deploy override extensions created using the Designer Workbench.

Also, a JSB can be used if the base screen is launched through a JSB or through a JSB that uses a JavaScript library to render screens.

Note: UNIX/Linux file paths are used in the following procedure.

---

1. Install the application and build a WAR file for it.
   a) Deploy the WAR file on the server in the exploded format.
   b) Start the application server by passing the following argument:

   ```
   -Dwufdevmode=true
   ```

2. Make sure that the changes made using the Extensibility Workbench have all the Java files generated and saved.
3. In the <*INSTALL_DIR*>/extensions folder of your installation directory, create the following subdirectory:

   <*application package name*>/webpages

4. In the webpages subdirectory, replicate the directory structure of the screen that you want to extend (relative to your deployment) and copy in all of the script files generated by the Extensibility Workbench.

   For example, if you extend the Manage Flight Route screen (which uses the file path <*application package name*>/flightRoute), you would copy all of the extension Java files into the <*INSTALL_DIR*>/extensions/<*application package name*>/webpages/<application package name>/flightRoute directory.

5. Create a new JSB file in the same folder to launch these newly generated files. The <ExtensionJSFile>_overlays.js files should be included before the corresponding <ExtensionJSFile>.js

files. You can use the JSB template of the Code Template Generator to create the code for this file. You would have to paste the code into the new file.

Sample code for JSB:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="scuiIDE"
            author="Sterling Commerce Pvt.Ltd">
    <target name="flight_route"
<!-- The name attribute in <target> is used to uniquely identify
this JSB in the application. It serves as its identifier.-->
            file="/extn/stk/flightRoute/test-all.js"
            loadAfter="flightService"
<!-- The loadAfter attribute in <target> is used to specify the
javascript library after which the current JSB should be rendered.-->
            allowDynamicLoad="true"
            debug="True"
            shorthand="False"
            shorthand-list="">
        <include name="/extn/stk/flightRoute/test_overlays.js"/>
        <include name="/extn/stk/flightRoute/test.js"/>
    </target>
</project>
```

6. Create a new servlet to register the new JSB file. The extn folder should be prefixed for LoadJSLibraryXml and loadIncrementalMashupExtnXml calls.

   Sample code for creating the servlet:

```
package jsbCreator;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import
com.sterlingcommerce.ui.web.framework.helpers.SCUIJSLibraryHelper;
import
com.sterlingcommerce.ui.web.framework.helpers.SCUIMashupHelper;
public class CreateServlet extends HttpServlet {
        private static final long serialVersionUID =
4693417985837892469L;
        public synchronized void init(final ServletConfig config)
throws ServletException {
                //loads the JSB specified at the path
                SCUIJSLibraryHelper.loadJSLibraryXml
("extn/stk/flightRoute/test.jsb", config.getServletContext());
                //loads the mashup XML specified at the path
                SCUIMashupHelper.loadIncrementalMashupExtnXml
("/extn/stk/flightRoute/test_mashup.xml", config.getServletContext());
        }
}
```

7. Package the servlet into a jar file.
8. Update the web.xml file with your customizations.
9. Relaunch the application to display the extended changes.

The changes will appear overlaid on the base screen.

Compiling and Minifying JavaScript Files in the Web UI Framework

1. Run the jscompile command to get possible JavaScript compilation warnings using the sci_ant.sh command from the <Install>/bin directory. This command works with the jsUtil.xml file in the same directory. This command can include the following properties:

   Note: This is an optional step and not a requirement for minification.

   - gis.install: Installation directory path.
   - srcDir: Source directory.
   - errorOnly: Indicates whether to check for all warnings and errors (false) or for errors only (true). Defaults to **false**.
   - format: Output format - (h) for html/(t) for text. Defaults to **t**. If errorOnly is set to true, only html (h) is the valid option.
   - outputFile: Output file path. If file path is not provided or file doesn't exist. all warnings will be directed to standard output.
   - warningOptions: Warning options (comma separated). Default options: [onevar, undef, forin, debug, browser, eqeqeq, newcap, evil]. For all warning options, see http://www.jslint.com/

   For example:

   ```
   ./sci_ant.sh –f jsUtil.xml jscompile –Dgis.install=<Install Dir>
   –DsrcDir=<Install Dir>/repository/eardata/platform_uifwk/<version>/war/platform
   ```

   Note: If you are using sci_ant.sh, then gis.install becomes optional.

2. Combine your files into one file by minifying the files using the sci_ant.sh command from the <Install>/bin directory. This command works with the jsUtil.xml file in the same directory. This command can include the following properties:

   - gis.install: Installation directory path.
   - jsbDir: JSB directory path (mandatory).
   - minify: Indicates whether files should be minified (true/false). Defaults to **true** (minify files). Optional.
   - srcDir: Source directory. Will be used if input attribute is not specified in JSB. Optional.
   - destDir: Destination directory. Will be used if input attribute is not specified in JSB. Optional.
   - createIndividualFile: Indicates whether to create individual files (true/false). Defaults to **false** (do not create individual files). Optional.
   - jscompile: Indicates whether to get JavaScript warning/errors (true/false). Defaults to **true** (get errors).

   For example:

   ```
   ./sci_ant.sh –f jsUtil.xml minify-js –Dgis.install=<Install Dir>
   –DsrcDir=<Install Dir>/repository/eardata/platform_uifwk/<version>/war
   –DjsbDir=<Install Dir>/repository/eardata/platform_uifwk/<version>/war/builder
    –DdestDir=<Install Dir>/repository/eardata/platform_uifwk/<version>/war
   ```

where <version> is either **20** or **30** depending on if you are using Ext JS 2 or Ext JS 3 JavaScript-related files/content.

Note: If you are using sci_ant.sh, then gis.install becomes optional.

If minification is required for extended JavaScript files, you should create an extn folder within the directory where overlays/extensions are added (<install-dir>/extensions/<application name>/webpages). Copy all of the files to be minified to that directory. You must follow the process of creating the same relative directory structure for extensibility. You can then run the minification script successfully because the minified file path in the JSB file does exist.

When you run the buildear/buildwar script, the following happens:

1. First, all contents of the overlays/extensions directory except the extn directory are copied to the <application war>/extn directory.
2. Then, the contents of the extn directory in the overlays/extensions directory get copied to the <application war>/extn directory. As the contents of this directory are copied last, it would override the contents contributed by overlays/extensions directory in case of a conflict (same directory structure).

Customizing web.xml in the Web UI Framework

1. Run the buildear or buildwar utility to create the EAR/WAR file.
2. Copy the web.xml.sample file from the *<INSTALL_DIR>*/repository/eardata/<package-name>/extn directory to the same directory with the file name "web.xml".
3. Modify the newly created web.xml files as needed.
4. If you need to add a new servlet or filter, package it in a jar file and run the *<INSTALL_DIR>*/bin/install3rdParty.sh script to include this jar file in a classpath.
5. Run the buildear or buildwar utility to create the EAR/WAR file.

Changing Bundle Files in the Web UI Framework

You can change bundle files in one of two ways:

- Through localization.
- Through extensibility.

1. If you are changing a bundle file through localization, you must replicate the folder structure of your current bundle file in the localization folder of the application.

   For example, if your bundle file is at /folder1/folder2/x-bundle.js and you are localizing or replacing a bundle entry for the fr-FR locale, then you should create a bundle file with the new values for the bundles that you want to change and retain all existing values at /localization/fr/FR/folder1/folder2/x-bundle.js.

2. If you are changing a bundle file through extensibility, do the following:
   a) Create your bundle files which only have the bundle entries that you want to replace.
   b) Identify the target name of the JSB that is being used to render the screen whose bundles should be replaced. The name should be entered in the loadAfter attribute of your JSB.

c) Specify only the path and name of your bundle-js file in the extn directory in the tag <include name>. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="scuiIDE"
            author="Sterling Commerce Pvt.Ltd">
    <target name="flight_route"
<!-- The name attribute in <target> is used to uniquely identify this JSB in
 the
application. It serves as its identifier.-->
            file="/extn/stk/flightRoute/test-all.js"
            loadAfter="flightService"
<!-- The loadAfter attribute in <target> is used to specify the javascript
library
after which the current JSB should be rendered.-->
            allowDynamicLoad="true"
            debug="True"
            shorthand="False"
            shorthand-list="">
        <include name="/extn/stk/flightRoute/flightRouteList_bundle.js"/>
    </target>
</project>
```

Designer Workbench of the Web UI Framework for Custom Developers

The Designer Workbench allows you to use WYSIWYG tools to build new screens for the application. It has tools similar to the Extensibility Workbench, which is used to change the screens of an out-of-the-box installation of the application.

Custom developers access the Designer Workbench by clicking the **Design new screens** link in the Extensibility Workbench.

Note: Although you can access the Designer Workbench from an out-of-the-box installation of the application, you are limited in the changes that you can make. Please contact Sterling Commerce Customer Support when changing an out-of-the-box installation using the Designer Workbench.

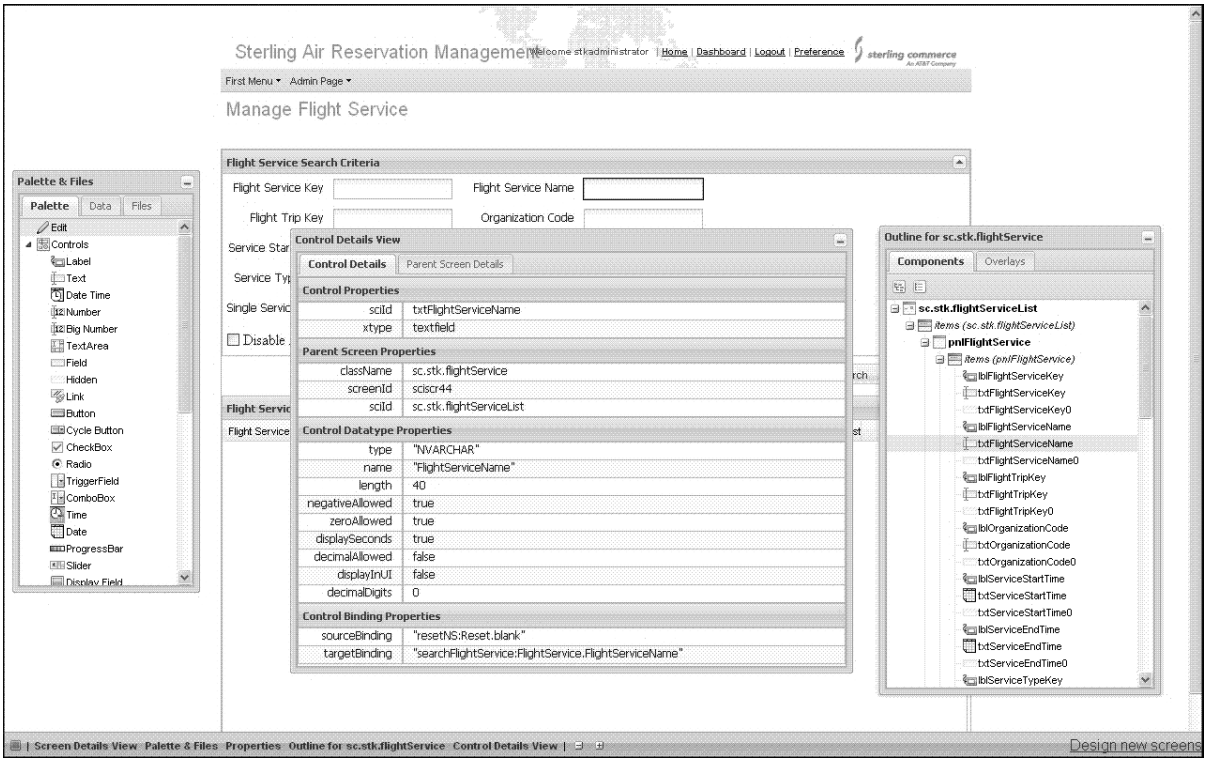
Control Details View of the Web UI Framework

The Control Details View of the Extensiblity Workbench displays basic control/parent screen details and information like bindingData and datatype information. It shows some additional information that is not present in the Original Properties grid.

The contents of the Control Details View are updated according to the selected control. The view does not expand and collapse with all of the other Extensibility Workbench views, because of the large amount of space that it uses.

The Control Details View includes two tabs:

- Control Details (default active tab)
- Parent Screen Details



The Control Details tab shows detail about the selected control under the following categories:

- Control Properties

  Shows the basic information about the selected control like sciId and xtype. The remaining config properties are viewable under the Properties View. sciId is shown to uniquely identify the selected control. In case of a column, the dataIndex is also displayed. If the column has an editor, then the editor xtype is also shown.

- Parent Screen Properties

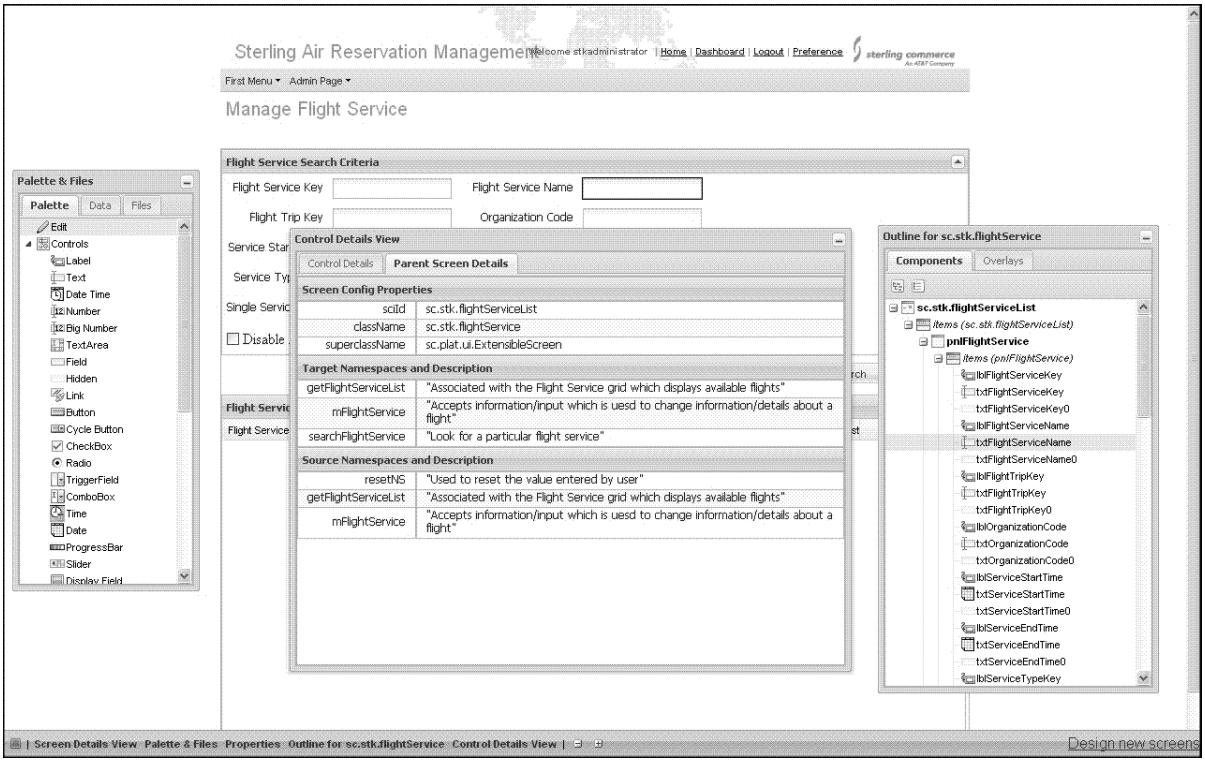
  Shows information that can be used to uniquely identify the control's parent screen.

- Control Datatype Properties

  Displays information about the datatype which is computed from the binding or scuiDataType attribute(provided during screen development).

- Control Binding Properties

  Displays all the control bindingData properties. For example, if the selected control is a textfield, then source and targetBinding would be displayed. In case of a combo, in addition to source and targetBinding the optionsBinding would also be displayed(provided, these properties have been defined on the control during development).



The Parent Screen Details tab includes information about the className, superclass, sciId, namespaces, and namespace description.

- Screen Config Properties

  Displays screen information like sciId, screen class and superclass name.

- Target Namespaces and Description

  Displays target namespaces (target bindings if namespaces not available) and their corresponding description from the namespacesDesc attribute. If no description is provided, the value is left blank.

- Source Namespaces and Description

  Displays source namespaces (source bindings if namespaces not available) and their corresponding description from the namespacesDesc attribute. If no description is provided, the value is left blank.

Note: If there are no values available for Control Datatype properties, Control Binding properties and Source/Target Namespaces and Description, then an empty panel with the same name would be populated.

Property Restrictions in Extensibility in the Web UI Framework

Certain properties should not be added during extensibility. If you add any of these properties from the Properties View, then you would get a relevant message in the console (Mozilla Firefox only) and the property would not be added in the Properties View. The property list is as follows:

- All controls: defid, id, _original_sciId, sciId, xtype.

- text, number, bignumber, textarea, triggerfield, combo, time, date and spinner (ext 3 only): (restricted properties listed in all controls) + vtype.
- containers and their sublcasses: (restricted properties in all controls) + items
- panel and its subclasses: (restricted properties in all containers) + tbar, bbar, buttons
- gridpanel and subclasses: (restricted properties in panels) + columns
- screen: (restricted properties in panels) + className, classId, superclassName, regXtype

Note: bindingData should not be added/modified for any base screen control. You can add/edit it if the control is added during extensibility.

Note: Grid supports the extn_bindingData property, which can be used during extensibility (if the grid is a base screen component) to add new fields to the Grid's store.

Adding Namespaces to Screens Using Extensibility in the Web UI Framework

1. Select the screen in the Tree View.
2. Type **namespaces** in the Properties View.
3. For every source and target namespace added, you need to provide a name and description. It is recommended that the names of all namespaces added during extensibility begin with extn_ to easily identify them.

   Note: The methods getTargetModel(), getModel(), and setModel() can be called from the extension and would return results from the combined model of the screen and extension.

Building and Customizing Pages/Controls with the Web UI Framework

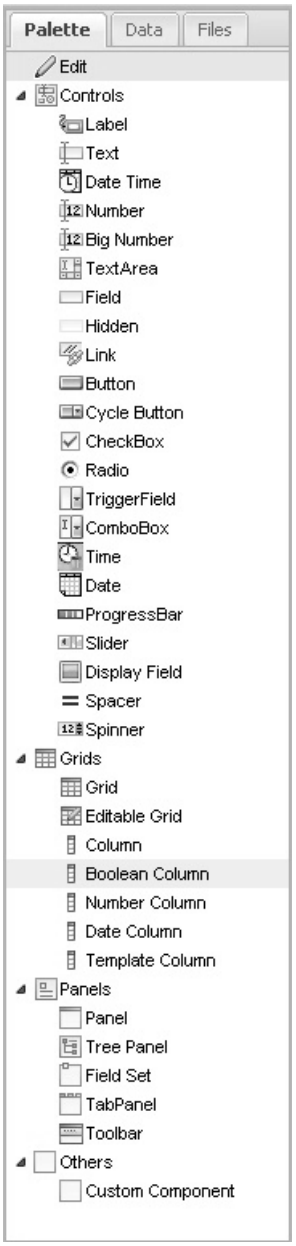
Widgets of the Web UI Framework

The following tables describe the widgets that are available in the Palette view of the Designer Workbench and the Extensibility Workbench. For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

Most of the widgets can be used with both Ext JS 2.2.1 and Ext JS 3.0. Widgets that can be used with only one version are identified. The graphic of the Palette shows the widgets available in Ext JS 3.0.

The following items cannot be created with widgets:

- Menus. Work with the Ext.menu.Menu class of Ext JS.
- Wizards. Wizards must be defined using an XML file.
- Repeating panels with radio buttons. Work with the Ext.form.Radio class of Ext JS.



Controls

Note: For information about the Column widget, refer to the Grids section. This widget appears in the Controls section in Ext JS 2.2.1 and in the Grids section in Ext JS 3.0.

Control	Description
Label	Text that can identify other controls.
Text	Text for input and display.
Date Time	Date and time values.
Number	Number values, up to 16 digits.

Control	Description
Big Number	Number values, more than 16 digits. The Big Number control works in the same way as the Number control, except that the getValue() function returns a string. The Big Number control is a customized control that is not documented in the online Ext JS documentation. It extends the features of the Text control.
TextArea	Multiple lines of text.
Field	Provides default event handling, sizing, value handling, and other functionality.
Hidden	Hides values in forms that need to be passed when you submit a form.
Link	Connects outside of current screen (for example, a URL).
Button	When clicked, causes an action. **Note:** Ext JS 3.0 does not support the Toolbar.Button, Toolbar.Splitbutton, or the menu button.
Cycle Button	Button that cycles through menus as you click it. Down arrow displays items for each menu.
Checkbox	Single box that flags a value as true or false.
Radio	A type of checkbox that can be grouped with other radio controls and allows only one control in a group to be checked.
TriggerField	A type of text field that includes a clickable trigger button.
ComboBox	A type of text field that includes a list of values from which you can select.
Time	Time input field with a time dropdown tool and automatic time validation.
Date	Date input field with a date picker tool and automatic time validation.
ProgressBar	Shows progress of an operation.
Slider	Supports vertical or horizontal orientation, keyboard adjustments, configurable snapping, axis clicking, and animation.
Display Field (3.0 only)	Display-only text field that is not validated or submitted.
Spacer (3.0 only)	Provides a gap or a space in a layout.
Spinner (3.0 only)	A number field with up and down arrows to increase or decrease the value in particular increments.

Grids (3.0 only)

Grid/Column	Description
Grid	Panel that includes table-like columns and rows.
EditableGrid	A type of grid that allows cell editing on selected columns.
Column	Normal/default column for a grid.
Boolean Column (3.0 only)	Renders boolean data fields.

Grid/Column	Description
Number Column (3.0 only)	Renders numeric fields according to a format string.
Date Column (3.0 only)	Renders date fields according to a specified format.
Template Column (3.0 only)	Renders values by processing a record's data using the XTemplate.

Panels

Note: For information about the Grid and EditableGrid widgets, refer to the Grids section. These two widgets appear in the Panels section in Ext JS 2.2.1 and in the Grids section in Ext JS 3.0.

Panel	Description
Panel	Container that can include: • Bottom and top toolbars • Separate header, footer and body sections • Built-in expandable and collapsible behavior • Prebuilt tool buttons that can be customized
Tree Panel	Tree-structured representation of hierarchically organized data.
Field Set	Groups form fields.
TabPanel	Groups tabs which can respond in unique ways to being activated and de-activated.
Toolbar (3.0 only)	Container that can include virtually any type of component.

Others

Use the Custom Component option in the Others section to add, to the screen, a component that you have created (like another screen developed using the Designer Workbench). The Custom Component option gives one screen access to a second screen whose components include that first screen (child-to-parent screen access). Use the preset properties to specify this access.

Working with Widgets in the Web UI Framework

Use the Outline View (Extensibility Workbench) or the Tree View (Designer Workbench) to do the following when you are working with widgets:

• Re-arrange widgets

Different rules apply to how you can re-arrange widgets. For example, you cannot move a button from a standard panel to a standard grid. The Designer Workbench has built-in safeguards against the improper re-arrangement of widgets. An error message appears if you try to move a button from a standard panel to a standard grid. These safeguards help you organize your screen in the most functional way.

You can re-arrange widgets in the following ways:

• Change the order of widgets. For example, you can place a text field between two buttons.

- Move a widget from the main screen to a panel, or from one panel to another panel. For example, you can move a button from a standard panel to a field set. Or you can move a column from a standard grid to an editable grid.
- Move one standard panel onto another standard panel.

If a screen uses a special tab sequence, re-arranging widgets might affect your intended sequence of actions when you tab from one widget to another.

- Delete widgets

  You cannot delete a widget from the base/out-of-the-box screen. You can only delete widgets that were added using the Extensibility Workbench. If you delete a widget that was used to work with data (like a text box or combo box), you will have to find another way to work with that data.

- Select a widget so that you can view or change its properties in the Properties view. If you are unsure of what widget is being shown in the Outline or Tree View, click the widget on the screen, and a horizontal blue line will show the location of the widget in the view.

  You can always use the canvas to select a widget, but it is not as precise as using the Outline View or Tree View. On the canvas, it might take you several clicks to select the right widget.

  For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

Adding a Widget to a Screen with the Web UI Framework

1. Make sure the Palette tab is showing on the Extensibility Workbench or the Designer Workbench, and not the Data or Files tab.
2. Click on a widget.
3. (Required only if a preset is required) Click the checkbox for the preset.
4. (Required only if a preset is required) Make any changes to the default properties of the widget by creating a preset property.
5. Move your cursor to the screen and right-click or left-click to add the widget to the screen. A tooltip will help confirm when you can add the widget.

   In the Designer Workbench, the widget appears in two places:

   - On the canvas, in the most upper left hand location.

     For example, if you add a button to a blank canvas, it appears in the upper left hand corner of the canvas. If you add a second button, it appears directly beneath the first button.

   - In the Tree View, in a hierarchical pattern under the screen object.

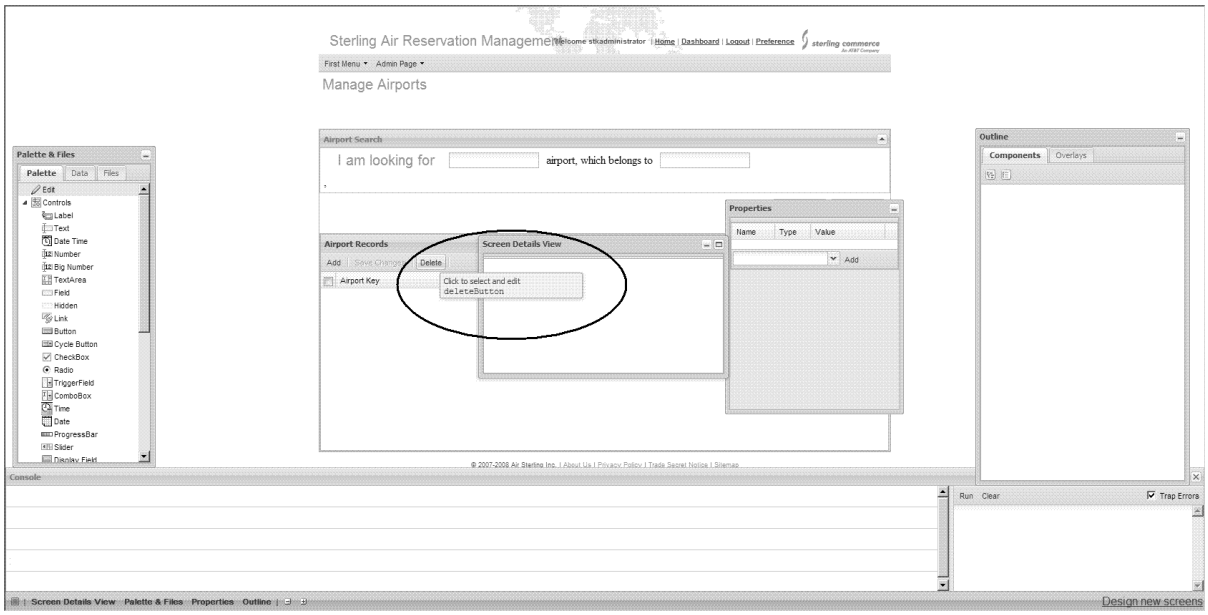
   In the Extensibility Workbench, the widget appears in two places:

   - On the screen, in the location where you clicked.
   - In the Outline view, in a hierarchical pattern under the screen object.

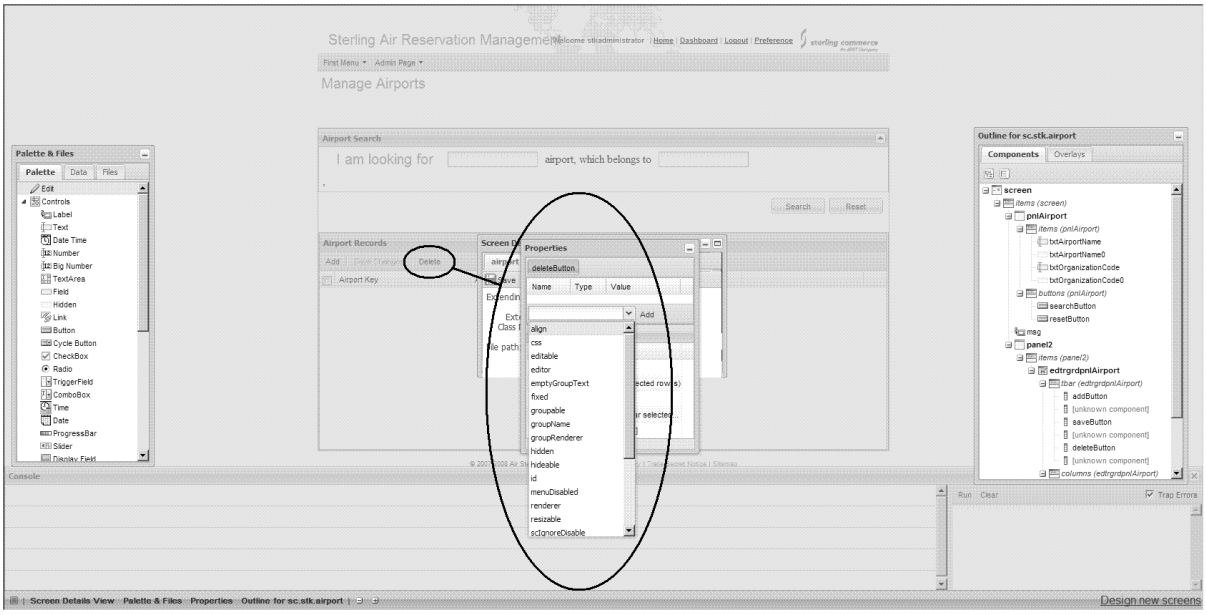
Customizing Widgets in an Existing Installation with the Web UI Framework

1. Open an existing installation of the application.
2. Navigate to the screen where you want to work.
3. Click **Shift + space bar** to bring up the Extensibility Workbench.
4. Select the widget that you want to modify. In the following example, the tooltip shows that you can select the **Delete** button.

Note: The button/links present in a base screen will continue to function unless you provide your action/click handler in the Properties View and click on **Refresh Instances** to apply those changes (If you directly click on a button/link on the screen, you might lose your changes if the action redirects you to another page). Hence, it would be best to select and edit these controls through the Tree View instead of the screen.



5. Make any changes to the widget, using the tools of the Extensibility Workbench. In the following example, you can add a property to the **Delete** button using the Properties view.



6. Save your changes.

Hiding Fields with the Web UI Framework

1. In the Designer Workbench or the Extensibility Workbench, add a panel to the screen.
2. Set the hidden property of the panel to **true**.
3. Add to the panel the field that you want to hide.
4. Save the screen.

Accessing the Working Files of the Web UI Framework

Access the working files in your project (*.json screen files) using the Files tab.

The Files tab includes the following tools:

- The **Options** button enables you to specify the project directory where you store the project files.
- The button with three dots near the top of the tab enables you to display the contents of a different directory.
- The **Notify project** checkbox near the bottom of the tab enables you to immediately update your main project with changes that you make using the Designer Workbench.

Viewing Screen Objects in the Outline or Tree View of the Web UI Framework

Use the Outline view (Extensibility Workbench) or the Tree View (Designer Workbench) to collapse and expand all or part of your list of screen objects. Collapsing and expanding the list does not affect the screen.

You can do the following:

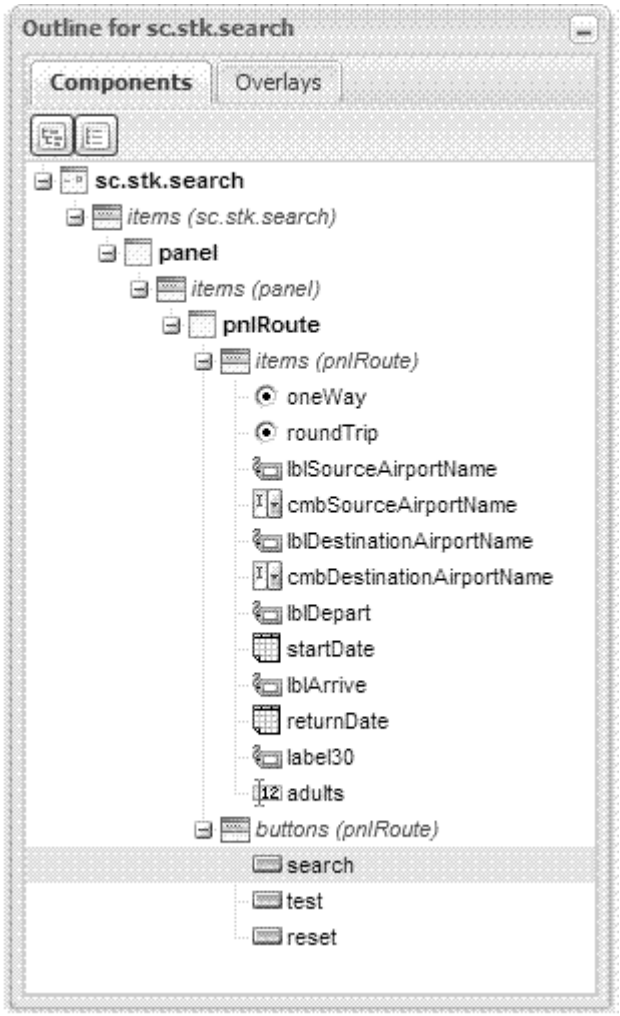
• Collapse the list so that only the screen object appears.

  If you collapse the list, and then click on the plus sign for the screen object, only the first level of widgets appear. This gives you a more general view of the screen.

• Expand the list to show all of the widgets.

If you are unsure of what widget is being shown in the Outline or Tree View, click the widget on the screen, and a horizontal blue line will show the location of the widget in the view.

Outline view (example for a **Search** button):



Tree View (draft screen with panel, buttons, and field):

Configuring Properties for Screens, Widgets, and Other Items with the Web UI Framework

Use the Properties view in the Designer Workbench to work with the properties of the canvas widgets. The Properties view settings work in conjunction with the settings in the Configure Properties dialog box.

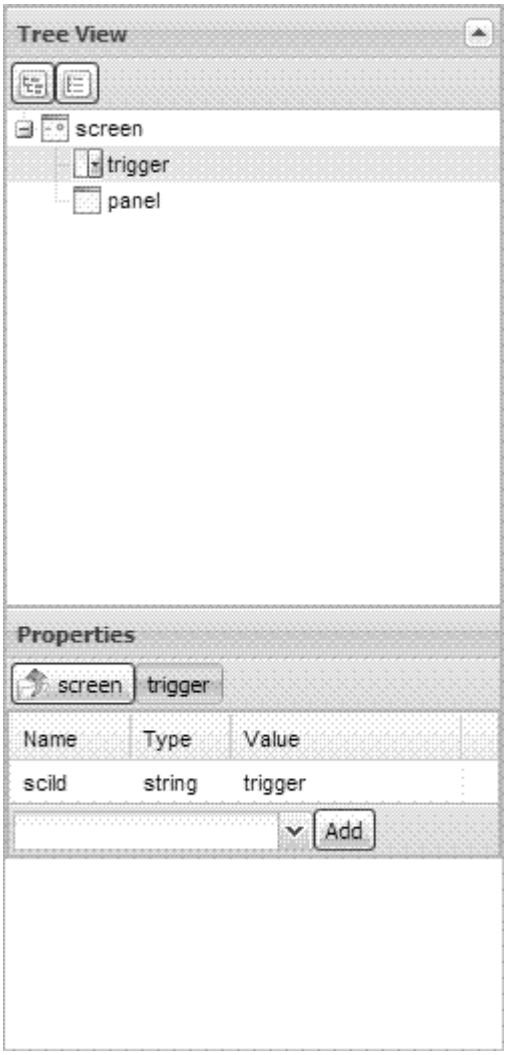
For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

1. Select that widget in the Tree View.

   Note: The button/links present in a base screen will still continue to function unless you provide your action/click handler in the Properties View and click on **Refresh Instances** to apply those changes (If you directly click on a button/link on the canvas, you might lose your changes if the action redirects you to another page). Hence, it would be best to select and edit these controls through the Tree View instead of the canvas.

2. Make sure that the tab for that widget is showing in the foreground of the Properties view, and not the tab for the screen (which always shows in either the foreground or the background).

   For example, if you select a trigger widget in the Tree View, tabs will show for both the screen and the widget.



3. To change the type or value of an existing property, click in the Type or Value field for that property, and make your change.

   For example, for a text field, you could change the Type from **string** to an expression (**expr**), and then enter an expression in the Value field.

4. Add a property.
   a) Use the down arrow by the **Add** button to select a new property. The dropdown list shows all of the available properties for that widget that are not default properties.
   b) Click the **Add** button.

      The property appears with the other properties. The default type for that property appears in the Type field.

   c) If necessary, change the default type to an expression (**expr**).
   d) Enter a value for the property.
   e) Change the property name to a unique name.

For example, for a button, you could add an enableToggle property, leave its type as boolean, and set its value to **true**.

5. Continue working with the screen.

Providing Description Attributes for Binding Namespaces in the Web UI Framework

The namespaces attribute on a screen provides information about the namespaces that have been used for binding controls. Customizers can view the namespaces and their descriptions (if provided by the developers) in the Control Details View of the Extensibility Workbench or the Properties window in the Designer Workbench. They do not have to open the JavaScript source files to view the namespaces.

The namespaces attribute is an object with two attributes (source and target) which individually are arrays of an object. In older screens, this attribute is an object with two attributes (source and target) which individually are arrays of a **string**. You can use the Designer Workbench to update the namespaces structures of older screens to use object arrays instead of string arrays.

The following shows the namespaces structure:

```
namespaces: {
     source: ['a', 'b']
     , target: ['c', 'd']
}
, namespacesDesc: {
     sourceDesc: ['description for a', 'description for b']
     , targetDesc: ['description for c', 'description for d']
}
```

The namespaces are saved under the namespaces object while the descriptions are saved under the namespacesDesc object. There is a one-to-one mapping between the contents in namespaces and namespacesDesc. This maintains backward compatibility with older applications that used a different namespaces structure.

An older screen that uses the different namespaces structure is upgraded in the Designer Workbench and the new structure is written to the JavaScript and JavaScript Object Notation files. The user receives a prompt, and on confirmation the update finishes. Once updated, the file can no longer be opened in an older version of the Designer Workbench. The new contents are written into the js and json files once the user saves them using the **Save** button.

After updating, the description field for both the source and the target will default to, respectively, **Description for <source name>** and **Description for <target name>**.

With older screens where namespaces were not provided in the Designer Workbench and not present in the js and json files, this upgrade generates the following js file:

```
namespaces: {
     target: []
     , source: []
}
, namespacesDesc: {
     targetDesc: []
```

```
     , sourceDesc: []
}
```

With these kinds of screens, you should provide a one-to-one mapping between the bindingData source and target arrays with namespacesDesc. In extensibility, if namespacesDesc has non-empty values while namespaces is empty, values would be displayed corresponding to bindingData.

Wizards of the Web UI Framework

Wizards guide users through the steps of a task in a specific sequence. Wizards are required for complex or infrequently performed tasks where the user is unfamiliar with the steps involved.

You must use an XML file to create the flow definition of your wizard. You cannot use widgets on the Palette tab of the Designer Workbench to create wizards. Also, you cannot use the Extensibility Designer to customize wizards. After you create a flow definition, you must register it.

Once a wizard is created, all client/server communication must be Ajax-based.

The wizard flow definition includes the following items:

- Page

  The visible part of the wizard. Each page must have a unique ID within the wizard.

- Rule

  Determines the flow of the wizard. A rule can lead to:

  - The next page in the wizard flow.
  - Another rule.

- Transition

  A connector which connects the wizard flow together. A transition can happen from page-to-page, page-to-rule, rule-to-page, or rule-to-rule.

- Flow controller

  The flow controller drives the wizard flow, and does the following:

  - Determines the next wizard entity that is shown or evaluated, based on the current activity entity.
  - Provides basic navigation capabilities like showNextPage and showPreviousPage.
  - Tracks data to be remembered in a session when a page transition occurs.

  Any wizard controller has to extend ISCUIWizardFlowController. The default wizard controller class is com.sterlingcommerce.ui.web.platform.wizard.SCUIDefaultWizardController.

  The wizard flow controller utility class orchestrates the flow based on the definition. You can plug in your own flow controller or use the default flow controller.

- Breadcrumbs

  Multiple pages can be grouped into the same category, which allows for the logical grouping of pages and the reduction of steps shown in the breadcrumb.

  You can add your own breadcrumbs to the application using utilities provided on the front end. The sc.plat.ui.Wizard class contains all the utilities.

Creating a Wizard with the Web UI Framework

Do the following in the flow definition XML file:

1. Specify the following attributes in the wizardEntities tag:
   - Page
   - Rule
   - Transition
2. Specify the flow controller attributes in the wizard tag.

   If you leave it blank or undefined, com.sterlingcommerce.ui.web.platform.wizard.SCUIDefaultWizardController will be used as a controller.
3. Specify the breadcrumbs attributes in the categories tag.
4. Specify the Web UI Framework front end of the wizard.

   On the UI, a wizard should be extended from the sc.plat.ui.Wizard class. A Wizard is a container with specific functionalities related to wizards. A Wizard can contain wizard pages (instances of sc.plat.ui.WizardPage or sc.plat.ui.ExtensibleWizardPage). With its card layout. a Wizard can switch between multiple wizard pages.

   The sc.plat.ui.Wizard class extends the sc.plat.ui.Screen class, which adds data binding capabilities to it.

   The doAction method, doBreadcrumbAction method, and other methods have been provided in the class for navigation.

   This class also fires various events which can be used to redraw a breadcrumb panel or a navigation panel.

   For more information refer to the JavaScript documentation for these classes.

Wizard Page Attributes in the Web UI Framework

The following table shows the page attributes to specify in the wizardEntities tag of your XML file when you create a wizard:

Attribute	Description	Constraints
id	The ID of the wizard page.	Should be unique within the wizard.
impl	The JSP/Struts action which renders the page.	Mandatory
type	Indicates the type of entity (PAGE or RULE).	Mandatory
start	Indicates if this page is the starting entity of the wizard.	Only one PAGE or RULE should be marked as true.
last	Indicates if this page is the last entity of the wizard.	Only one PAGE or RULE should be marked as true.
category	Indicates the category of this page. Used for breadcrumbs.	
namespace/name	The namespace for which data would be sent out of this page. If a rule originates from this page, the namespaces	

Attribute	Description	Constraints
	should be a superset of the defined namespaces for that rule. There can be many such namespaces.	

Wizard Rule Attributes in the Web UI Framework

The following table shows the rule attributes to specify in the wizardEntities tag of your XML file when you create a wizard:

Attribute	Description	Constraints
id	The ID of the wizard rule.	Should be unique within the wizard.
impl	The rule implementation: • Java class Implements a predefined interface. • greex Returns a string output.	
type	Indicates the type of entity (PAGE or RULE).	Mandatory
start	Indicates if this page is the starting entity of the wizard.	Only one PAGE or RULE should be marked as true.
last	Indicates if this page is the last entity of the wizard.	Only one PAGE or RULE should be marked as true.
output/value	The allowed output from the rule. There can be many such outputs.	
namespace/name	The namespace for which data would be sent to the rule. If this rule originates from a page, the namespaces should be a superset of the defined namespaces for that page. There can be many such namespaces.	

Wizard Transition Attributes in the Web UI Framework

The following table shows the transition attributes to specify in the wizardTransition tag of your XML file when you create a wizard:

Attribute	Description	Constraints
id	The ID of the transition.	Should be unique within the wizard.
source	The ID of the source entity from which this transition originates.	Value should be same as ID of one of the defined PAGE or RULE types.

Attribute	Description	Constraints
target	The ID of the destination entity at which this transition ends.	Value should be the same as the ID of one of the defined PAGE or RULE types.
output	Required when the originator of this transition is a rule. The target is chosen based on the output which the rule calculates.	
output/value	The output of the rule.	
output/target	The ID of the destination entity.	Value should be same as ID of one of the defined PAGE or RULE types.

Wizard Flow Controller Attributes in the Web UI Framework

The following table shows the flow controller attributes to specify in the wizard tag of your XML file when you create a wizard:

Attribute	Description	Constraints
id	The ID of the transition.	Should be unique within the webapp.
flowController	The fully qualified class name of the flow controller.	If the attribute is empty or does not exist, the controller provided by the framework will be defaulted.
independentPages	Indicates if the wizard pages are independent of each other (true or false). If this is set to false, when a previous page is shown, all pages until the requested page are discarded. Data for the discarded pages is lost. However, you can set up an event that saves the discarded pages.	Defaults to false.
finishImpl	A URL which handles the wizard's save action.	

Wizard Breadcrumb Attributes in the Web UI Framework

The following table shows the breadcrumb attributes to specify in the categories tag of your XML file when you create a wizard:

Attribute	Description	Constraints
category.id	The ID of the category.	Should be unique within the flow.
category.description	The resource bundle key.	Mandatory
category.style	The css class that needs to be applied to the breadcrumb.	Optional

Sample XML Flow Definition for Wizards in the Web UI Framework

Each page of a wizard needs to specify namespaces it can allow as output from that page. If a rule is invoked after a page, these namespaces should be a superset of the namespaces defined for that rule.

```
<wizard id="<application>.sampleWizrd"
flowController="com.sterlingcommerce.ui.web.platform.wizard.SCUIDefaultWizardController"
independentPages="false">
<wizardEntities>
  <wizardEntity id="Page0"
  impl="/<app_dir>/wizard/wizardpage1.jsp" last="false" start="true"
  type="PAGE" category="SampleWizard.category0">
    <namespace name="ns1">
    <namespace name="ns2">
    <namespace name="ns3">
  </wizardEntity>
</wizardEntities>
    <wizardTransitions>
       <wizardTransition id="NewTransitionId6" source="Page0" target="Rule"/>
       <wizardTransition id="NewTransitionId9" source="Page1" target="Page3"/>
       <wizardTransition id="NewTransitionId10" source="Page2" target="Page3"/>
       <wizardTransition id="NewTransitionId7" source="Rule">
           <output target="Page1" value="1"/>
           <output target="Page2" value="2"/>
       </wizardTransition>
    </wizardTransitions>
    <categories>
         <category id="SampleWizard.category0"
description="category0" style="simple"/>
         <category id="SampleWizard.category1"
description="category1" style="simple"/>
         <category id="SampleWizard.category2"
description="category2" style="simple"/>
    </categories>
</wizard>
```

Preset Properties in the Web UI Framework

You can configure the properties of a Web UI Framework widget so that it has preset properties whenever you create a new instance of it. This feature overrides the default properties of the widget.

For more information about widget properties, refer to the Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.

The Designer Workbench includes two types of presets:

- Out-of-the-box presets

  These presets are shipped along with the Designer Workbench and are provided for a few widgets. They are not editable.

Although out-of-the-box presets are read-only, their contents can be copied, and used to create new editable presets.

- Custom (user-defined) presets

  You can define any number of custom presets for every widget, provided each one has a unique name. These presets are stored in your project directory in the designer-metadata folder.

You can also work with presets in the Extensibility Workbench, but you have fewer options than you have with the Designer Workbench.

With the Designer Workbench, you can work with presets in the canvas and in the following views:

- Palette
- Data
- Tree

In the Palette and Data views, the presets selected for a widget serve as the default preset until the browser is refreshed or any other preset is selected from the list. Any widget dropped on the canvas is initialized with the properties of the selected preset.

In the Tree View and on the canvas, selecting a preset for a widget results in the addition of the properties of the preset to that widget. When applying a preset to a widget, no existing properties are deleted. Any new properties from the preset are added to the widget. Preset properties that already exist in the widget are updated.

With the Extensibility Workbench, you can do the following with preset properties:

- Right-click on a widget in the Palette & Files View and select a preset to apply.
- Create new presets.

With the Extensibility Workbench, you cannot do the following with preset properties:

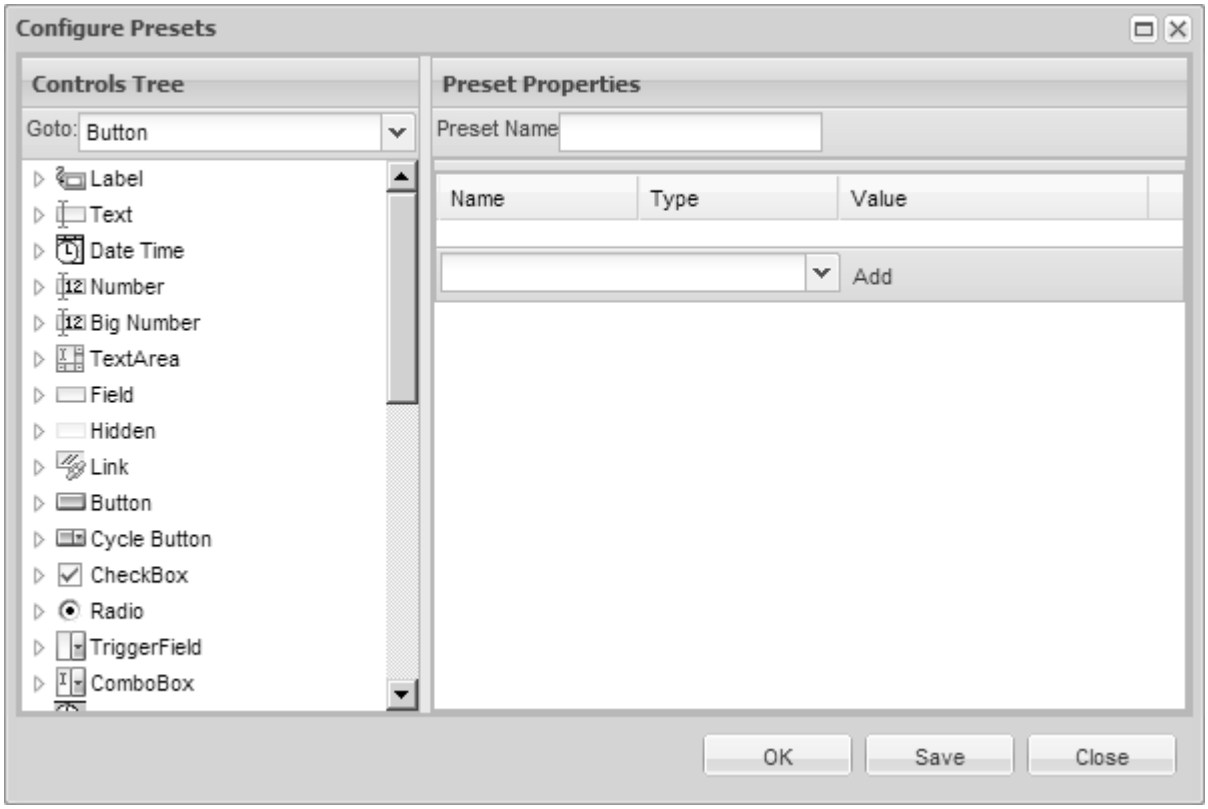
- Apply a preset to a control using the Outline View.
- Apply a preset to a control by right-clicking on the screen.

Creating Preset Properties with the Web UI Framework

1. Open the Designer Workbench.
2. In the Palette or Data Sources view, right-click any item and select the **Configure Presets** option.

   The Configure Presets dialog box appears, with any preset properties for all of the widgets.

   You can work with the presets of any customizable widget, and not just the widget that you right-clicked in the Designer Workbench to display the Configure Presets dialog box.



The Controls Tree view on the left side contains the widgets that have:

- Out-of-the-box properties that you cannot change, but which you can copy to make a new widget.
- Properties that can be customized with new presets.

When you work with new preset properties, you can save individual presets by clicking the **Save** button. You can save all of the presets at once by clicking the **OK** button, which also closes the Configure Presets dialog box.

3. To create brand new presets for a widget, do the following:
   a) Right-click a widget in the Controls Tree view.
   b) Choose the **create new preset** option.

      A default name for the preset appears.

   c) Select the new preset.
   d) In the Preset Properties view, use the default name or type a new name for the preset in the Preset Name field. Click the **Save** button to save a new name in the Controls Tree view.
   e) To create a brand new property for a preset, type the name of the new property in the combo box adjacent to the **Add** button, and click the **Add** button. A line for this property appears in the Preset Properties view. In the Type field, you can make this new property a string value or an expression. In the Value field, enter the string value or expression. Click the **Save** button to immediately save this new property, or click the **OK** button to save all custom properties at the same time and also close the dialog box.
   f) To create a property that is based on an existing property (like maxlength or labelStyle), select that property from the combo box adjacent to the **Add** button, and click the **Add** button. A line for this property appears in the Preset Properties view. In the Type field, you can make this new property a value

of its default data type (number, boolean, etc.) or an expression. In the Value field, enter a value or expression. Click the **Save** button to immediately save this new property, or click the **OK** button to save unsaved presets and their properties at the same time and also close the dialog box.

4. To create a preset for a widget that is based on another preset of that widget, do the following:
   a) Right-click a preset in the Controls Tree view.
   b) Choose the **create from selected** option.

      A copy of that preset is created.

   c) In the Preset Properties view, type a new name for the new preset property in the Preset Name field. Click the **Save** button to save the name in the Controls Tree view.
   d) To create a preset property for a brand new property, type the name of the new property in the text box with the down arrow, and click the **Add** button. A line for this property appears in the Preset Properties view. In the **Type** field, you can make this new property a string value or an expression. In the **Value** field, enter the string value or expression. Click the **Save** button to immediately save this new property, or click the **OK** button to save all custom properties at the same time and also close the dialog box.
   e) To create a property that is based on an existing property (like maxlength or labelStyle), select that property from the combo box adjacent to the **Add** button, and click the **Add** button. A line for this property appears in the Preset Properties view. In the **Type** field, you can make this new property a value of its default data type (number, boolean, etc.) or an expression. In the **Value** field, enter a value or expression. Click the **Save** button to immediately save this new property, or click the **OK** button to save all custom properties at the same time and also close the dialog box.
5. When you are finished, and you have not closed the Configure Presets dialog box, do one of the following to save preset changes.

   A preset can contain a number of preset properties. A preset is a holder for all the properties that a user enters. When you click the **Save** button, a preset is saved.

   • Click the **Close** button. Only presets that you have saved using the **Save** button will be kept.
   • Click the **OK** button. Any preset changes will be kept.

Applying Preset Properties with the Web UI Framework

1. Right-click on an existing or newly created widget on the canvas or in the Tree View.

   The Apply Preset menu appears.

2. Move your mouse over the Apply Preset menu to display your preset options (you do not have to click). If preset properties are configured for that widget, you will see the **Default Properties** option, an out-of-the-box preset (if provided), and at least one preset option. If no preset option appears, you will need to create one.
3. To apply a preset option, select that option. To display the properties for that option in the Properties view, re-click on that widget to refresh the Properties View with the new set of properties.
4. If you want to go back to the default properties for that widget, right-click the widget and select the **Default Properties** option.

Enabling a Child Screen to Access a Parent Screen with the Web UI Framework

A screen can contain another screen within it. If the child screen requires access to the parent screen, you can use a property in the child screen to access any component or property in the parent screen.

1. Click on the Palette tab in the Designer Workbench.
2. Right-click on the Custom Components option in the Others category.
3. Check the box by Parent Handle, which is an out-of-the-box preset provided with the Designer Workbench.
4. Move your cursor over the canvas until you see the tool tip **Click to add customct in screen**.
5. Right-click or left-click where you want to add this access handle.
6. Make any property adjustments to the handle. You cannot change the default property values of either scOwnerScr (**this**) or xtype (**panel**) in the Configure Presets View. These default properties indicate a handle from a child screen to a parent screen.

   You can create other presets for this control.

   Once you select this preset for a Custom Component and add the widget to the screen, you can view these two properties in the Properties View below the Tree View. You can modify the property values or add new properties, since you are now working on an instance of a Custom Component that was added to the screen (and not on the preset) and that was initialized with your values.

Menu Customizations with the Web UI Framework

You can make menu customizations using the Ext.menu.Menu class of the Ext JS JavaScript framework.

The Designer Workbench does not have a widget for creating menus. However, you can create menus using the menu template of the Code Template Generator, which you access from the Designer Workbench using the **Generate Code** button. Code generated by the Code Template Generator is pasted or typed on the Code Update page, where you finish the creation of the code for the new menu. For more information, refer to the documentation for the Code Template Generator.

In Web UI Framework, to show a menu in a screen, you have to get menu data from the server and render it on the browser.

In the default implementation, the tag includeMenu is provided, which can be called from JSP as:

```
<scuiimpltag:includeMenu></scuiimpltag:includeMenu>
```

This returns all of the menus configured for the logged-in user for which the user has permissions.

This tag returns menu data as JSON data (which can have text), a URL, JavaScript, or an image.

```
{
 text: 'First Menu',
 subMenu: [{
 text: 'First SubMenu',
 url: '/<app_dir>/<app_dir>/editRule.do'
 js: 'openpopup()',
```

```
 img: 'my-cls-img'
}
```

To render this data, the default implementation is provided as a JavaScript file. To use this file, include the following code in the JSP:

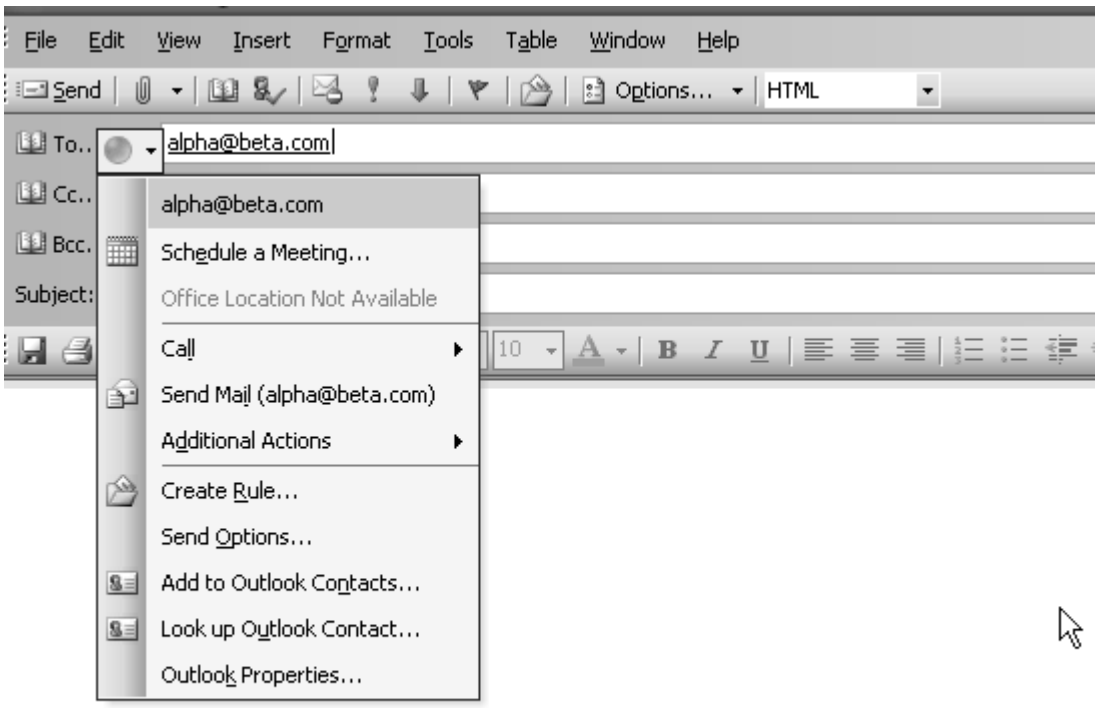
```
<script type="text/javascript"
src="<%=request.getContextPath()%>/platform/scripts/menuPaint.js"></script>
```

Use the following guidelines for menu customization:

- To change the UI look and feel of the menu, it has to use its own implementation instead of the application menuPaint.js.
- To get menu data with more information, it has to use its own implementation instead of the application using includeMenu tag.

Creating Smart Tags with the Web UI Framework

Smart tags are used to recognize certain types of data. For example, when you hover your cursor over a component which has a smart tag, a list of actions that can be performed are displayed. The following graphic shows an example of a smart tag:



To use a smart tag with a component, do the following:

1. Register the smart tag actions for a component. Use the class sc.plat.SmartTagActionRegistry to add action providers to the registry, using the following methods:

   • registerActionProvider(obj, boolOverride)

   A valid action provider object must contain a getActions method that accepts a reference of the type Ext.Component. It must also contain an “id” property that is the unique ID of this action provider object. The getActions method must return an array of objects that can have the following properties:

   • categoryid: The unique ID of the category object.
   • sequenceid: Sequence number which helps in sorting.
   • item: Config of Ext.menu.Item

   The following is an example of an object that can be returned by the getActions method. This example uses the default ID category (**DEFAULT**).

```
{
   categoryid: 'DEFAULT',
        sequenceid: 1
        item: {
        text: 'Show a Ext.Window',
            handler: function(){
            new Ext.Window({
               width: 600,
               height: 500
              }).show();
                }
              }
}
```

   • registerActionType(name, id)

   Registers the type with the action provider object corresponding to the ID passed. Before doing this, you must first register an action provider object with that ID using the registerActionProvider method.

   The default UI displays the actions returned. You can use the default UI or override it by registering your own UI provider.

2. Set the scSmartTag property for that component to one of the following values:

   • true

   The default implementation of the application gets the attribute in the source binding data. Objects that are registered with a key equal to the attribute in the source binding data are fetched.

   • A value such as the attribute in sourceBinding or targetBinding.

   Objects that are registered with a key equal to the scSmartTag property are fetched.

Generating Code from Templates with the Web UI Framework

You can use templates (instead of hand-coding) to generate code for the following components:

• Mashup APIs

- Struts actions
- JSB (JavaScript Builder)
- Resources
- Resource permissions
- Resource and permissions (combines resource and resource permission templates)
- Menus

The Code Template Generator helps save you development time. For example, you can use a code template to update and test a mashup or JSB file, instead of hard-coding the file and re-starting the application server.

Access the Code Template Generator from the **Generate Code** button of the Designer Workbench to create code from either default or custom templates. Templates include static (fixed) values, as well as variable values that you can change. Code is generated in a manner similar to the manner in which code is generated for js and config.js files.

After you create the code, you can update the code in a running application without stopping the server. To update the code, you must copy it from the Code Template Generator and paste it in the file of the component that you are updating. The Code Template Generator does not create a new file or add the generated code to a file.

The Code Template Generator is optional, but Sterling Commerce recommends that you use it to reduce your development time. You can copy and paste generated code into mashup, Struts, JSB, resource, resource permission, and menu files, instead of hand-coding those files.

The Code Template Generator is not used for extending the application. However, you can use it to test override extensibility changes made using the Designer Workbench.

A new template appears in the Create New Template window, which you access from the Code Template Generator window. If you are creating a new template from an existing template, the Create New Template window includes variables that can be used in the code template and reduce the number of required fields in the Code Template Generator. If you are creating a new blank template, the new template is empty.

Code Template Generator of the Web UI Framework

Access the Code Template Generator from the Generate Code option in the top toolbar of the Designer Workbench. The generator includes the following sections:

- Available Templates

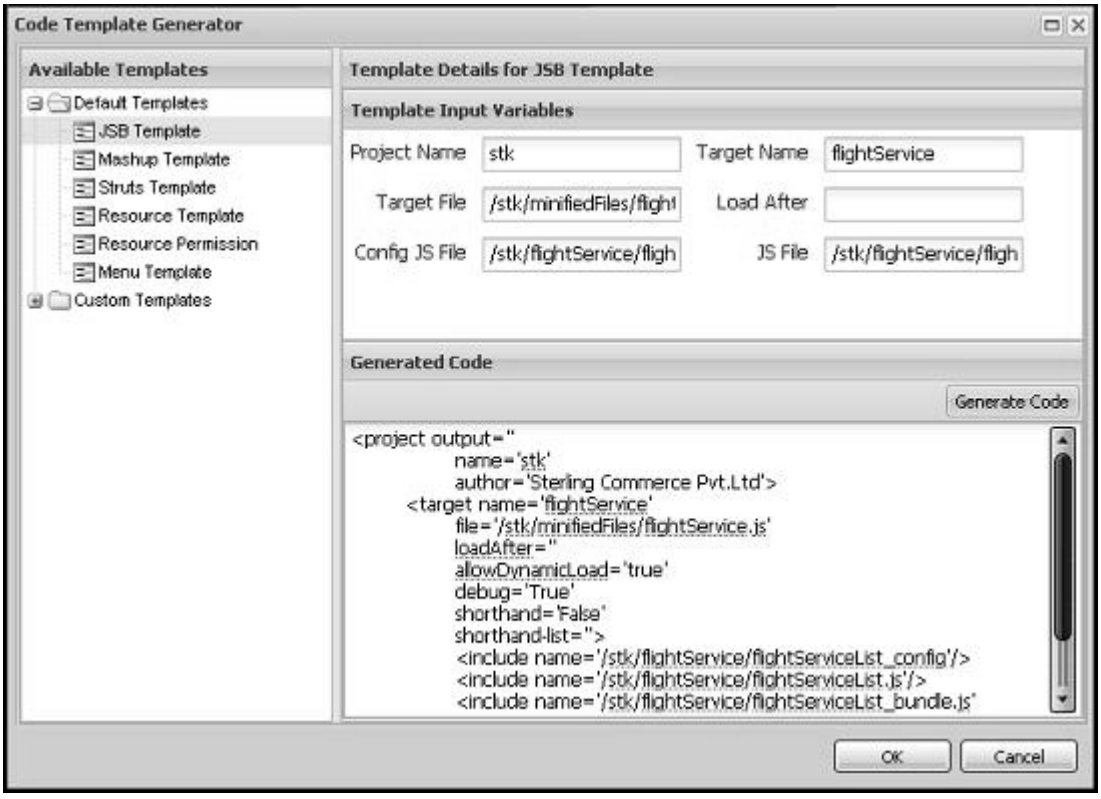
  Displays all of the available templates in a tree view. Includes default and custom templates. Default templates are available for JSB (JavaScript Builder), mashups, Struts actions, resources, resource permissions, and menus.

  The applications that only consume the uifwk clump would only have the default JSB template available in the existing Designer Workbench.

  Create custom templates from copies of the default templates.

- Template Details

  Includes variable text that you can change (the Template Input Variables section) and the code that is generated from the template (the Generated Code section). The fields in the Template Input Variables section depend on the type of template.



For example, you could create a new JSB template from the default template by right clicking on default JSB Template and then clicking on **Create from selected**. This brings up the Create New Template window. Then, you would use the User Defined Variables section of the Create New Template window to add a new target_name variable that uses the type expr and the value className. If you use the same code template, then you can remove the Target Name input as this variable would now get the value of the className variable. Once saved, this custom JSB template would have five input fields on the Code Template Generator window instead of the six input fields that appear on the default JSB template. You could also modify the template and use the className variable instead of the target_name variable, with the same result.

You cannot change a default template, but you can copy its contents to make a custom template. You can edit a custom template or copy its contents to make another custom template.

The fields that appear in the Template Input Variables section depend on the component that you are changing. The values that you enter in this section are assigned to the XML code that will be generated. These values are also stored in the json file of the screen.

For example, in the above graphic showing a JSB template, **stk** is the value of the Project Name variable, which is the label for the proj_name variable (JSB project) used in the code template. The **stk** value would be assigned to its corresponding variable (proj_name) and replaced in the generated code.

The values in the Template Input Variables section appear in the Generated Code section after you click the **Generate Code** button. You can directly change the generated code, but these changes will be lost if you click the **Generate Code** button again during this session. The text in the Generated Code field is not saved to a file on disk. You can copy the generated code to another file.

Clicking the **OK** button saves all of the screen's changes to the screen's json file and closes the Code Template Generator. That is, all of the variable values for all of the templates are saved. Clicking the **Cancel** button closes the Code Template Generator without saving changes to the screen's json file.

Default Code Templates of the Web UI Framework

The Web UI Framework includes the following templates that can be used to update different WUF screen components during runtime:

- Mashup APIs
- Struts actions
- JSB (JavaScript Builder)
- Resources
- Resource permissions
- Resource and permissions (combines resource and resource permission templates)
- Menus

WUF uses template metadata to create (by default) XML code templates, which in turn are used to generate XML code that is used by the application. However, you can define a custom template that is not an XML template.

Template updates occur in the following ways:

- In the current login session (all templates). The changes last only as long as you are logged in. You can test the changes while you are logged in.
- In the database (all templates except mashup and JSB). The changes are permanent unless you delete them. Updates of components in the database are an alternative to updates of those same components in the Application Manager.

The following sections describe how each template works and the XML elements for each kind of component.

Mashups

This updates only the mashup registry that is in the session (but not permanently in the database).

The generated template code can be used to update mashup.xml files, through copying and pasting. If you are using Aggregator mashups or adding more attributes to the mashup code, you must define your own template.

The following variables are used in the default code template for mashups:

Item	Description	Source
id	Uniquely identifies a mashup in the application.	User-entered.
description	Information about mashup.	User-entered.
apiName	API that is being used in mashup.	User-entered.
input	Input to mashup.	User-entered.
outputTpl	Output from server.	User-entered.

Struts

Struts can be updated by adding the generated code to the application's struts.xml file if:

- The struts.xml file is removed from the jar file in which it is packaged and placed under the WEB-INF/classes directory.

For example, in the case of stk, you would remove the stk_struts.xml file from the platform_ui_demo_app.jar file and place it under the WEB-INF/classes directory.

Any changes that you make to the struts.xml file would be reflected in the application. If two actions have the same ID, then the action that appears last in the file would be picked up in the application.

- The Struts dev mode is enabled (turned on automatically if the war file is built with -Ddevmode=true).
- The application is deployed as an exploded war file by using the following command:

```
buildwar.sh -Dwarfiles=<war file name> -Dappserver=<appserver> -Dnowebservice=true
 -Ddevmode=true
```

When using WebLogic 10, use the following command:

```
buildwar.sh -Dwarfiles=<war file name> -Dappserver=weblogic -Dnowebservice=true
 -Ddevmode=true Dwls-10=true
```

The following variables are used in the default code template for Struts actions:

Item	Description	Source
actionName	Uniquely identifies an action in the application.	User-entered.
id	ID of mashup used/called.	User-entered.
input_ns	Namespace used for input.	User-entered.
output_ns	Namespace used for output.	User-entered.
success_jsp_path	JSP to load if the action is successfully completed.	User-entered.

JSB (JavaScript Builder)

This updates only the JSB registry that is in the session (but not permanently in the database).

The following variables are used in the default code template for JSB (JavaScript Builder):

Item	Description	Source
proj_name	Name of the project.	User-entered.
target_name	Identifies a JavaScript library. Should be unique in the application.	User-entered.
target_file	Directory/file path for JavaScript library.	User-entered.
loadAfter	Specifies the library after which the current JSB library should be loaded.	User-entered.
config_js	Path of the config.js file for the screen.	User-entered.
js	Path of the JavaScript file for the screen.	User-entered.
bundle_js	Path of the bundle JavaScript file for the screen.	User-entered.

Resources

This updates or adds the resource in the current session and permanently in the database.

The following variables are used in the default code template for resources

Item	Description	Source
app_code	Application code.	User-entered.
app_name	Application name.	User-entered.
parent_resource_id	Resource ID of the parent resource.	User-entered.
resource_create_type	Resource create type (for example, **USER**).	User-entered.
resource_desc	Information about the resource.	User-entered.
resource_key	Unique key of the resource.	User-entered.
url	The action URL for the Struts action.	User-entered.

Resource Permissions

This updates or adds the resource permissions both in the current session and permanently in the database.

The following variables are used in the default code template for resource permissions:

Item	Description	Source
resource_perm_key	Resource permission key.	User-entered.
user_group_id	The ID of the group to which the user belongs.	User-entered.
user_group_key	The key of the group to which the user belongs.	User-entered.

Resource and Permission

This combines the resource and resource permission templates. It updates or adds resources and resource permissions both in the current session and permanently in the database.

The following variables are used in the default code template for resources and permissions:

Item	Description	Source
app_code	Application code.	User-entered.
app_name	Application name.	User-entered.
parent_resource_id	Resource ID of the parent resource.	User-entered.
resource_create_type	Resource create type (for example, **USER**).	User-entered.
resource_desc	Information about the resource.	User-entered.
resource_key	Unique key of the resource.	User-entered.
url	The action URL for the Struts action.	User-entered.
resource_perm_key	Resource permission key.	User-entered.
user_group_id	The ID of the group to which the user belongs.	User-entered.
user_group_key	The key of the group to which the user belongs.	User-entered.

Menus

This updates or adds a menu both in the current session and permanently in the database. It provides an alternative to the Application Manager method of creating a menu.

The following variables are used in the default code template for menus:

Item	Description	Source
menu_desc	Information about this menu entry.	User-entered.
menu_key	Key/ID used to uniquely identify this menu.	User-entered.
menu_type	The menu type.	User-entered.
parent_menu_key	The menu key of the parent.	User-entered.
resource_key	Key used to identify the source.	User-entered.

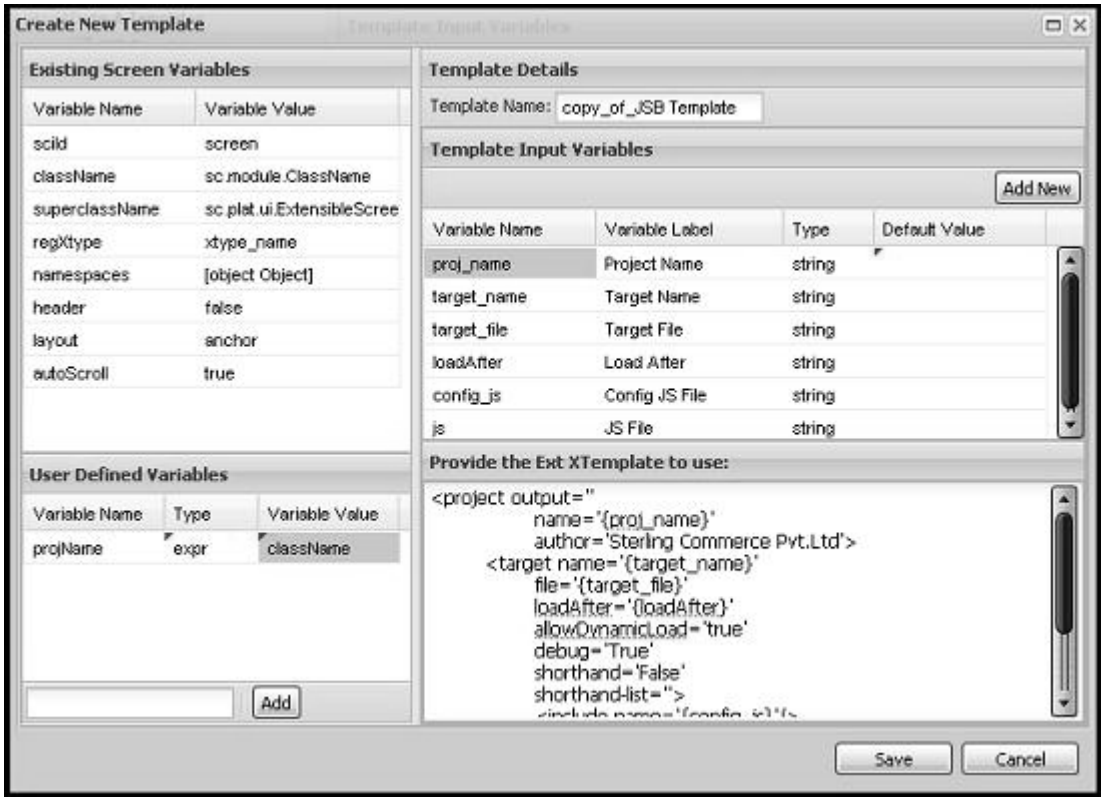
Creating a Custom Code Template with the Web UI Framework

Do the following to create a custom code template from either a default template or a custom template.

1. Open the Code Template Generator.
2. In the Available Templates section, right-click on a template in one of the following lists:
   - Custom Templates
   - Default Templates
3. Choose the **create from selected** option.

   A new template appears in the Create New Template window. This window includes variables that can be used in the code template and reduce the number of required fields in the Code Template Generator.

   The Existing Screen Variables section show the default variables, which you cannot change (names and values). The User Defined Variables section shows the variables that you created and which you can change (names, types, and values).



In the User Defined Variables section, you can add any properties to the name/value pairs that are part of the default template. Click the **Add** button to add a user-defined variable, which defaults to the **string** type. You can change the type to an expression (**expr**).

The above example screen shows a user-defined variable (projname) of the type **expr** and the value **className** (className is a screen variable). The application uses the expression projName=className, instead of the projName=”className” (if the type is **string**).

You can remove variables by right-clicking on a variable row and selecting the **delete** option.

4. In the Template Details section, define the following information:

   • Template Name: Unique identifier of a template that also serves as the json file name.
   • Variable Name: Name by which this parameter is referenced in the template.
   • Variable Label: Text that would be displayed before the input field of this variable on the Code Template Generator window.
   • Variable Type: String or expression (expr).
   • Value: String, variable, or expression that could be evaluated to return a value. This value would then serve as the default value and be used to populate the value in the input field for this variable on the Code Template Generator window. By default, this field is left empty.

   You can add variables by clicking the **Add New** button. A new variable row appears.

   You can remove variables by right-clicking on a variable row and selecting the **delete** option.

   The bottom part of the Template Details section shows the Ext JS XTemplate that is used to generate the code.

5. Save the new template by clicking the **Save** button. The new template is saved as a json file (using the template name) in the following directory:

   <user's project directory>/designer-metadata/templates

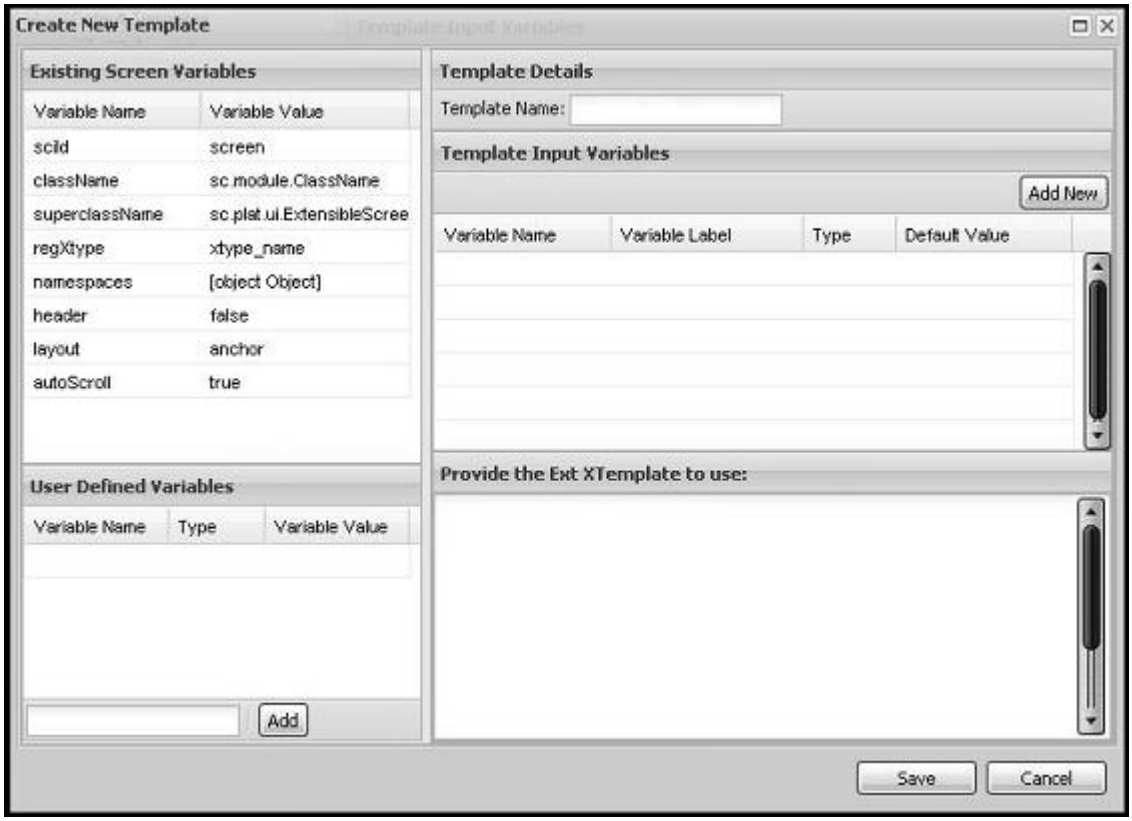
   If you do not want to save the template, click the **Cancel** button.

   After you click the **Save** button, the Create New Template window closes and the Code Template Generator window appears. You can now use the new template to update the code for a screen component.

Creating a Custom Code Template Using a Blank Template with the Web UI Framework

1. Open the Code Template Generator.
2. In the Available Templates section, right-click on the heading of the Custom Templates list.
3. Choose the **create new template** option.

   A blank template appears, with the default template name. All of the variable values are blank except for the variables in the Existing Variables section.



4. Add variables and enter values for them.
5. Click the **Save** button to create the Ext XTemplate for the new template. Click the **Cancel** button to exit the Create New Template window without saving any of your work.

   The Code Template Generator window re-appears.

6. Click the **Generate Code** button to create the custom template from the Ext XTemplate that you just created.

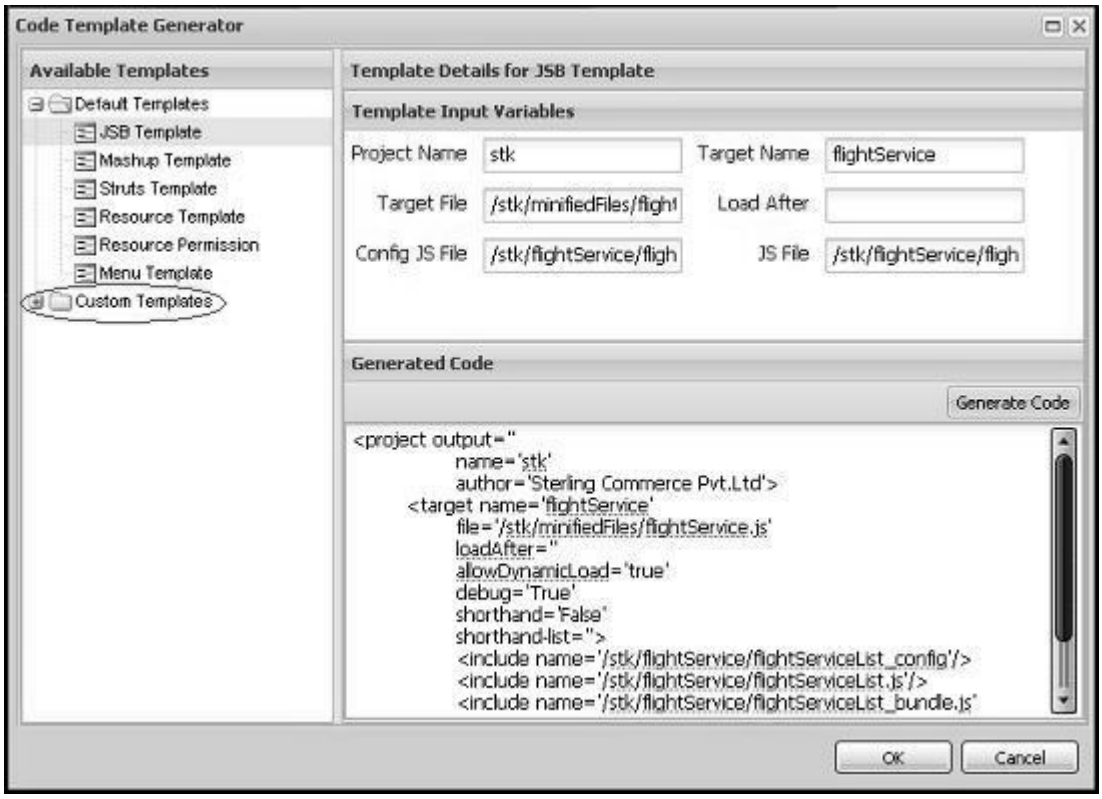
New code appears in the Generated Code section of the Code Template Generator window.

7. Save the custom template by clicking the **OK** button. You can exit the Code Template Generator without saving the custom template by clicking the **Cancel** button.

   After saving a custom template, it appears in the Custom Templates list in the Available Templates section of the Code Template Generator.

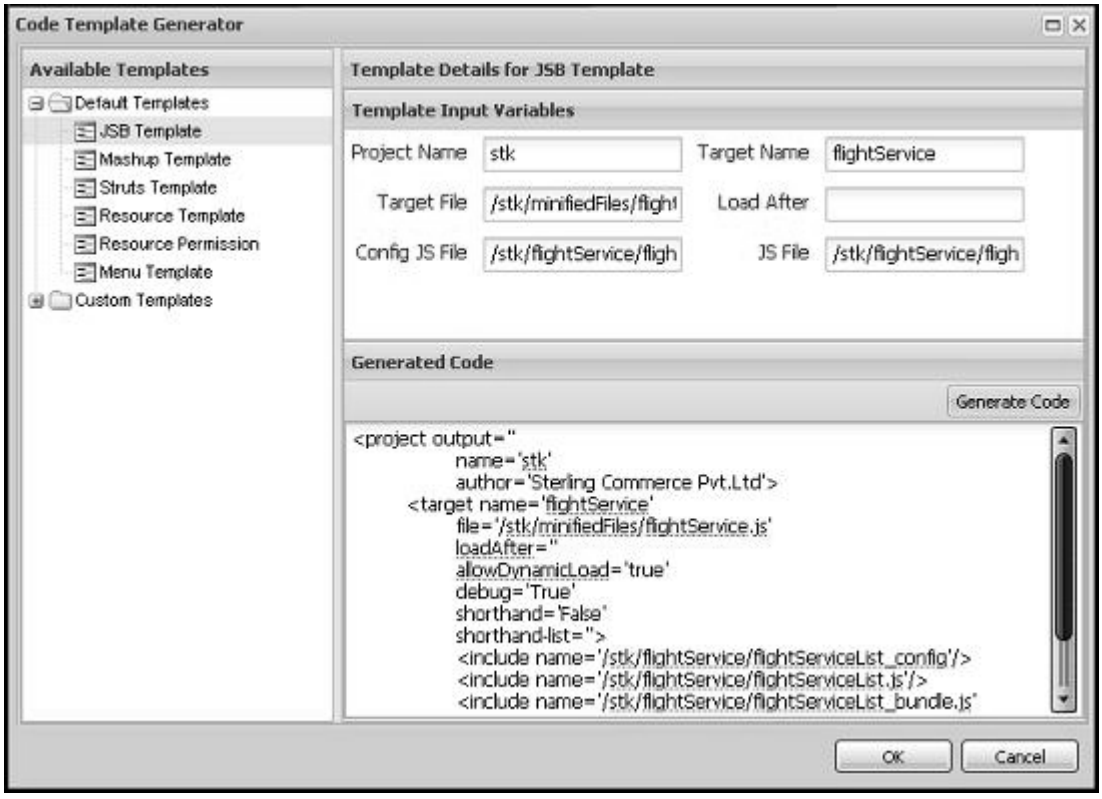
Editing a Custom Code Template with the Web UI Framework

1. Open the Code Template Generator.
2. In the Available Templates section, right-click on a template in the Custom Templates list.



3. Choose the **edit template** option.

   The template appears, with variables in the Template Input Variables section and the code in the “Provide the Ext XTemplate to use” section..



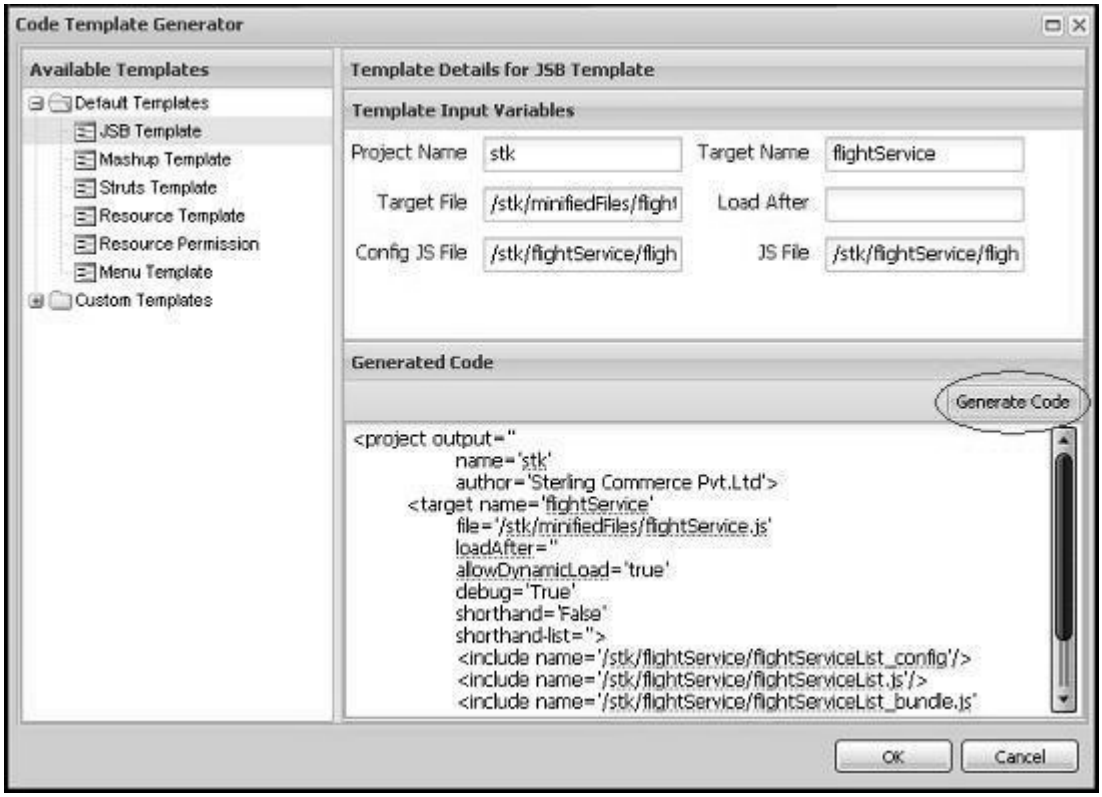
In the User Defined Variables section, update the values for any of the variables or add new variables.

4. Click the **Save** button to update the Ext XTemplate for the template. Click the **Cancel** button to exit the Create New Template window without saving any of your work.

   The Code Template Generator window re-appears.

5. Click the **Generate Code** button to update the template.

   New code appears in the Generated Code section.



6. Save the updated template by clicking the **OK** button.

   You can exit the Code Template Generator without saving the template by clicking the **Cancel** button.

Updating a Screen in a Running Application with the Web UI Framework

After updating the code of a screen component using a code template, you can update that component in a running application without stopping the server (hot code replace).

In the update window, you can paste in code or type in your own code.

You can use this procedure when you are adding new code or updating existing code. To override existing code, the new code must have the same resource ID as the existing code.

Hot code replace and code generation are completely independent of each other. Hot code replace enables you to view your changes in the application without having to update any files or restart the server. This reduces the total development time and also enables you to debug or test your changes to the application.

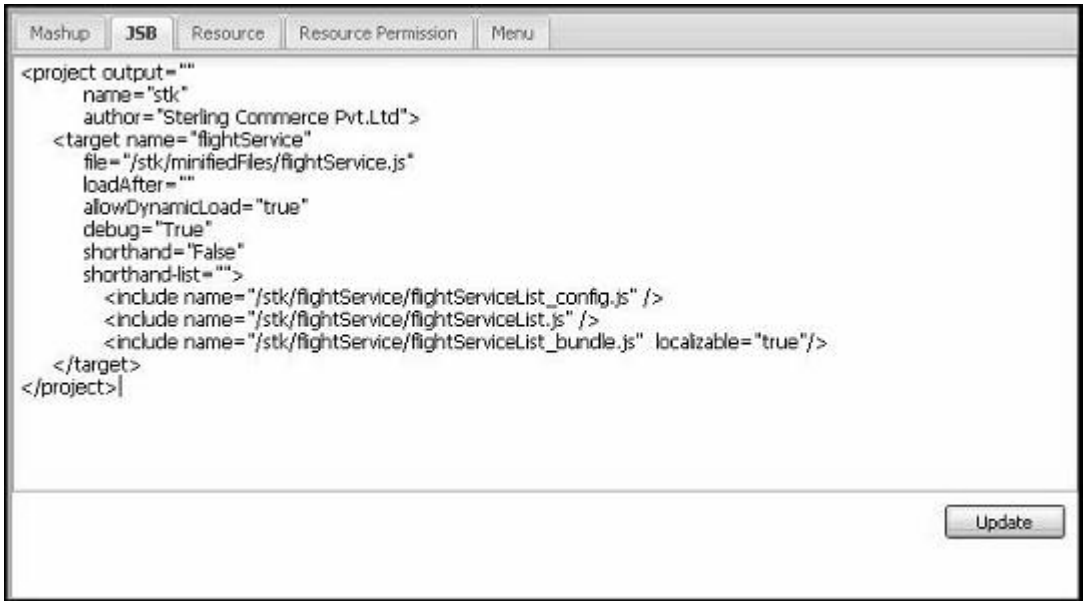
1. Make sure that you are running the application in Struts dev mode in exploded war mode (provide -Ddevmode=true while building the war file). Also, when launching the application, wufdevmode should be set to true.
2. Access the hot code replace screen using the following JSP:

   /platform/dev/afc_updatePage.jsp (In case of platform AFC)

   The Hot Code Update page for applications that consume the uifwk clump only would be at the following relative path:

   /platform/dev/uifwk_updatePage.jsp

To enable hot code replace for any other component, applications need to provide their own dev JSP and its implementation.



3. Click on the tab for the component that you want to update (Mashup, JSB, Resource, Resource Permission, Menu).
4. Paste in the code that you generated for that component in the Code Template Generator. You can also paste in code that you have hand-coded or directly type in code.
5. Update the application by doing one of the following:
   - To update one component, click the tab for that component and click the **Update** button.
   - To update more than one changed component (but not all changed components), click the tab for each component and click the **Update** button.

   Updating a resource, resource permission, or menu updates these components both temporarily in the current login session and permanently in the database. Updates to a mashup or JSB are made only in the current login session.

   When you press the Update button, the changes are written to the URL of the application, where you can test the changes.

Debugging Tools of the Web UI Framework

- Console

This appears at the bottom of the screen after you log in to the application. It includes the following features:

- Debug Console tab

  This shows the actions that you ran while tracing actions using the Start Trace button (for more information, see next bullet).

  It also includes a panel for your test scripts. You can trap errors using these scripts.

- DOM Inspector

  This shows the paths for files used on the screen (JavaScript, css, and other files).

- Start Trace button

  When you click this button, all of the actions that you take are recorded until you click the Stop Trace button. The button toggles between the Start Trace and Stop Trace labels. Your actions are recorded on the Debug Console tab of the Console.

  To display this button, click **Ctrl + F2**.

- View Screen Model button

  Clicking this button displays the Screen Model dialog box, which shows the following information:

  - Ext JS-based screen information.
  - The namespaces that are bound to this screen.
  - The data for these namespaces.

You can view this information in one of the following ways:

- Text View

  Shows the data in the JSON format.

- Tree View

  Shows property information about the object in a directory view.

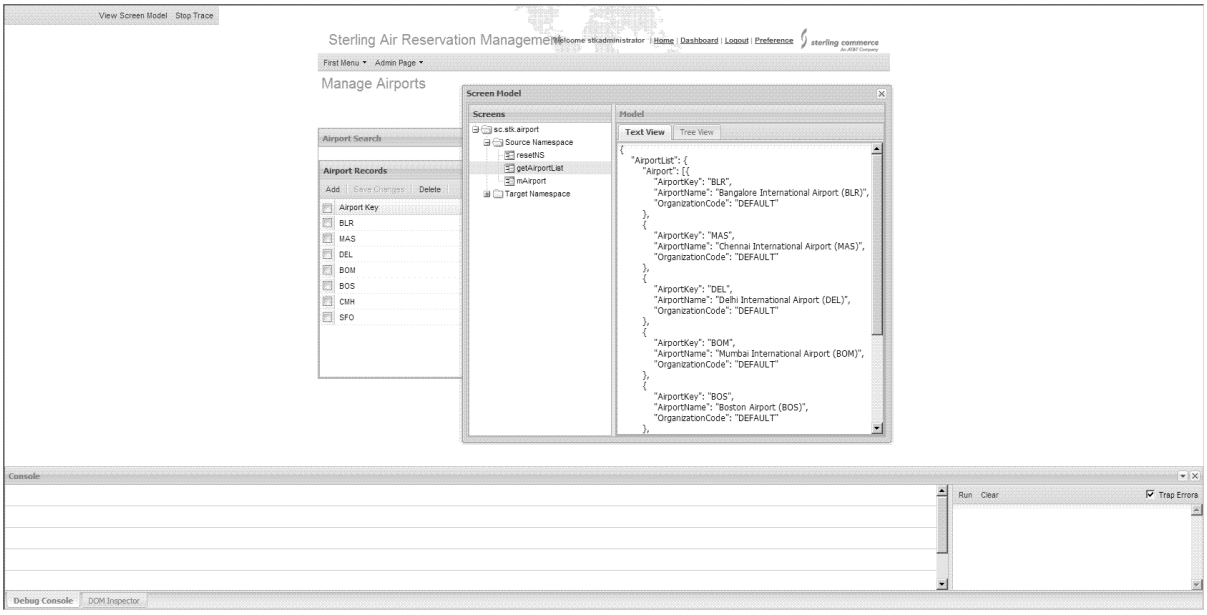
To display this button, click **Ctrl + F2**.

For an example of this button, do the following:

1. Click **Admin Page** > **Manage Airport**
2. Click the **Search** button.

   Search results appear.
3. Click **Ctrl + F2**.
4. Click the **View Screen Model** button.

   The Screen Model dialog box appears.



5. In the Screens panel, click down the tree through sc.[application].airport and Source Namespace and select the getAirportList namespace.
6. In the Model panel, click on the Text View and the Tree View to see different views of the data for the getAirportList namespace.

Setting Up Backend Logging in the web.xml File in the Web UI Framework

The Web UI Framework allows you to enable logging for backend framework usages (Struts, mashups, and API), based on the URI for each client. Once you have specified the logging, you can activate it using the **Start Request Log** button in the debugging toolbar in the application.

For each logging request, the WUF backend logging will be done for the following:

- When a Struts action is called, the Struts action name is logged.
- When a mashup is invoked, the XAPIMashup class name and the API name/Flowname are logged.
- When a redirect happens or requestDispatcher is created.

1. Use the scui-request-log-enabled context parameter of the web.xml file to set up backend logging. By default, this parameter is set to **true**. If it is not set to **true**, then the **Start Request Log** button does not appear.

   Sample web.xml entry:

```
<context-param>
  <param-name>scui-request-log-enabled</param-name>
  <param-value>true</param-value>
</context-param>
```

2. Use the com.sterlingcommerce.ui.web.framework.utils.SCUIUtils class to provide static methods to check if logging is enabled.

```
public static boolean isRequestLogEnabledInCtx(SCUIContext uiContext){
       return isRequestLogEnabledInCtx(uiContext.getWebContext().getRequest());
}
public static boolean isRequestLogEnabledInCtx(HttpServletRequest request){
        // read from context param if enabled
        // read from the boolean from isRequestLogEnabled
        return isRequestLogEnabledInCtx;
}
...
public static boolean isRequestLogEnabled(SCUIContext uiContext){
        return isRequestLogEnabled(uiContext.getWebContext().getRequest());
}
...
public
static boolean isRequestLogEnabled(HttpServletRequest request){
        // read from context param if enabled
        // read from the boolean from isRequestLogEnabled
        return isRequestLogEnabled;
}
```

3. Use the com.sterlingcommerce.ui.web.framework.context.SCUIContext class to actually log the message. The utility method in this class can be used by the application to log its request-based messages. It will be logged only if the scui-request-log-enabled context parameter is **true** and the user has started the request log via the debugging toolbar.

Note: A runtime exception is thrown if the method is called when logging is not enabled by, for example, another internal method.

```
...
public void setRequestLogMessage(String message){
 ...
 // check if log enabled and attribute already exists.
 this.setAttribute(SCUIConstants.REQUEST_LOG_MSG_PARAM_NAME, message);
}
...
```

4. A separate appender is used to put all request-based logging in a separate file. The requestinfo.log file is available at <install>/logs or your default log location.

   The log has to be enabled by one of the following actions:

   • The -Dyfs.logall=Y command
   • Enabling logging via the System Management Console.

   If the log is not enabled, no logging will take place.

   Sample log message:

```
2010-02-18 06:38:47,257:DEBUG  :[ACTIVE] ExecuteThread: '0' for queue:
'weblogic.kernel.Default (self-tuning)':
Inside SCUIAction flightTrip
Getting Request dispatcher /stk/flightTrip/flightTrip.jsp [system]: requestlogger
```

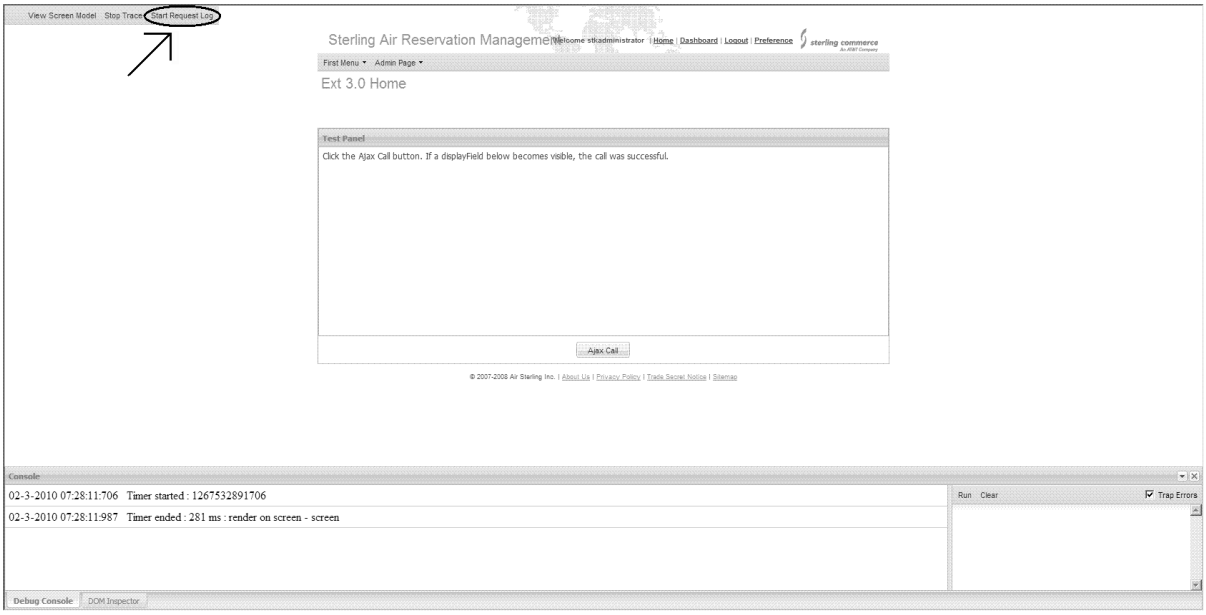
```
2010-02-18 06:39:08,806:DEBUG  :[ACTIVE] ExecuteThread: '4' for queue:
'weblogic.kernel.Default (self-tuning)':
Inside SCUIAction getFlightTrip
Inside SCUIXAPIMashup com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIMashup

Api name is getFlightTripList
Getting Request dispatcher /stk/flightTrip/gft.jsp [stkadmin]: requestlogger

2010-02-18 06:39:09,013:DEBUG  :[ACTIVE] ExecuteThread: '4' for queue:
'weblogic.kernel.Default (self-tuning)':
Inside SCUIAction getOrganizationList
Inside SCUIXAPIMashup com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIMashup
Api name is getOrganizationList
Getting Request dispatcher /stk/flightTrip/gol.jsp [stkadmin]: requestlogger

2010-02-18 06:39:11,833:DEBUG  :[ACTIVE] ExecuteThread: '4' for queue:
'weblogic.kernel.Default (self-tuning)':
Inside SCUIAction airport
Getting Request dispatcher /stk/airport/airportScreen.jsp [stkadmin]:
requestlogger
2010-02-18 06:39:15,836:DEBUG  :[ACTIVE] ExecuteThread: '4' for queue:
'weblogic.kernel.Default (self-tuning)':
Inside SCUIAction getAirportList
Inside SCUIXAPIMashup com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIMashup
Api name is getAirportList
Getting Request dispatcher /stk/airport/airportList.jsp [stkadmin]: requestlogger

2010-02-18 06:39:52,948:DEBUG  :[ACTIVE] ExecuteThread: '4' for queue:
'weblogic.kernel.Default (self-tuning)':
Inside SCUIAction activateRequestLog                        [stkadmin]:
requestlogger
```

5. The **Stop Request Log** button comes up on the debugging toolbar when the **Start Request Log** button is activated. Clicking the **Stop Request Log** button will stop the logging in the requestinfo.log file after a final Struts action is called.

Enabling Backend Logging in the User Interface with the Web UI Framework

1. In the user interface, display the **Start Request Log** button by clicking **Ctrl + F2**.

   The button appears in the debugging toolbar which appears in the upper lefthand corner of the screen.



2. To enable logging, click the **Start Request Log** button.

   The Console will not display any details about request logging. It will be available in the requestinfo.log file either in the default log directory or in the log directory specified by the user during installation.

   The button name changes to **Stop Request Log**.

   Log messages will be created when a Struts action is called, when a mashup is invoked, and when a redirect happens or a requestDispatcher is created.

3. To disable logging, click the **Stop Request Log** button. The request-based logging will stop.

   The button name changes to **Start Request Log**.

State Management in the Web UI Framework

The Web UI Framework provides a mechanism for state management, which enables the application to remember a user interface state and apply it across user sessions. You can implement state management with either the default implementation of the Web UI Framework or with a pluggable custom implementation. You can also customize the default implementation of the Web UI Framework.

The default Web UI Framework implementation of state management has the following features:

- The state is saved and restored from the database, where the PLT_USER_UI_STATE table contains the state information.
- The UI state is stored on a periodic basis. The time period is configurable.

Note: Saving the UI state on every change is performance-intensive.

- You can control the list of components whose state is stored/restored.
- State changes are remembered/cached. If there are no state changes, then the state is not saved.

- Utility methods are provided on both the client side and the server side to synchronously fetch the state. The UI state is cached on the client side once it has been accessed, to avoid multiple round trips to the server.
- If a particular user is deleted, the state for that user is automatically deleted from the PLT_USER_UI_STATE table.

Implementing State Management with the Web UI Framework

When you install the application that uses the Web UI Framework, a default implementation of state management is provided.

In the Ext JS JavaScript framework, Ext.state.Provider is the abstract base class for state provider implementations. This class provides methods for encoding and decoding typed variables, including dates and definitions of the Provider interface.

If you want to provide your own implementation of state management, the Ext.state.CookieProvider class has an example of this implementation.

The state is saved periodically by posting the data to a servlet, which in turn delegates the task to a class that is specified in the web.xml file. The UI framework defines an interface which the class in the web.xml file needs to implement.

```
<context-param>
    <param-name>scui-uistate-provider</param-name>
    <param-value>
     com.sterlingcommerce.ui.web.platform.state.SCUIStateProvider
    </param-value>
</context-param>
```

The default state provider works along with the Ext library in the following manner:

- Local caching of component states, for faster retrieval on subsequent actions.

Note: This might not be useful for multiple page applications.

- Saves the available/changed state into the database on page unload. This assumes that the session timeout has not occurred.
- The Ext library automatically calls the get state method for all components which are state-aware.
- The Ext library automatically calls the set state method whenever the state changes for a component.

By default, the state management implementation is not registered with the Ext JS library. The implementation needs to be registered by the application. This provides flexibility if the application does not require the UI state to be persisted.

Interface Contracts of the Web UI Framework - State Management on the Client Side and Server Side

The state management task has interface contracts on both the client side (JavaScript) and the server side (Java).

For more information, refer to:

- The Ext JS framework documentation at (2.2.1) http://www.extjs.com/deploy/ext-2.2.1/docs/ or (3.0) http://www.extjs.com/deploy/dev/docs/.
- The Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

Interface Contract	Description	Methods
SC.platform.state.StateProvider (client side)	Implements the Ext.state.Provider base class for state provider implementations. Ext.state.Provider has the following methods: • get Returns the current value for a key. • clear Clears a value from the state. • set Sets the value for a key. An example of a custom implementation of state management is in the Ext.state.CookieProvider class.	Includes the following utility methods: • Retrieves the state from the database, given the ScreenName and the ComponentName. • Retrieves a list of all component states from a database, given a ScreenName. • Persists a state to a database, given a ScreenName and a ComponentName. • Clears a state, given a ScreenName and a ComponentName.
ISCUIStateProvider (server side)	Manages the saving and retrieving of the UI state. Use one of the following methods to implement this contract: • Make the following web.xml context parameter entry: <param-name> scui-uistate-provider </param-name> <param-value> (Fully qualified class name of the implementation) </param-value> • Call the setUIStateProvider utility method of the SCUIStateHelper class.	• getUIState(userId, componentId, screenId, applicationId, uiContext) Retrieves the state of the given component. • getListOfUIStatesForScreen(userId, screenId, applicationId, uiContext) Retrieves the full list of state information for all components belonging to the specified screen. • init(servletContext) Performs initialization. Called only once in the life cycle. • saveUIState(uiState, uiContext) Saves/persists the provided state object. • saveUIStatesList(uiStateList, uiContext) Saves/persists the provided list of state objects.

Transaction Management in the Web UI Framework

The Web UI Framework provides tools for transaction management. This helps you decide how to start, end, commit, and roll back transactions, which ensures data integrity.

You can implement transaction management with either the default implementation of the Web UI Framework or with a pluggable custom implementation. You can customize the Web UI Framework implementation of transaction management. All customizations involve changes to the web.xml file and to the transaction management interface contract of the Base UI Framework.

Transaction management is handled in the mashup layer of the Web UI Framework. The mashup layer also handles authorization and connects the user interface of the application with the business logic (data layer). More than one mashup can be defined within the mashup layer, and one mashup is one transaction. If one mashup is nested within another mashup, the beginning and end of the parent mashup is one transaction.

Implementing Transaction Management with the Web UI Framework

You can implement transaction management in one of the following two ways:

- A custom implementation of transaction management that uses the interface classes of the Web UI Framework.

  Applications need to register their implementation for the ISCUITransactionContextFactory class either as a context parameter or by making a Java call to the method SCUITransactionContextHelper.setTransactionContextFactory.

  Registering can be done by either of the following methods:

  - Calling the static setter method SCUITransactionContextHelper.setTransactionContextFactory
  - Adding a context parameter scui-transaction-context-factory with the value as the implementation class name. In this case, the helper class will instantiate the context factory.

  ```
  <context-param>
        <param-name>scui-transaction-context-factory</param-name>
        <param-value>
         com.sterlingcommerce.app.TransactionContextFactory
        </param-value>
  </context-param>
  ```

  In both methods, the Web UI Framework will call the init method of the ISCUITransactionContextFactory class just after registering it. Applications can use this method do some initializations for the factory.

  Errors might occur if a factory class is not provided or if a class registered using the context parameter does not implement the interface ISCUITransactionContextFactory. In either of these situations, if a transaction context is requested, Web UI Framework will throw a SCUIException with the proper error message.

- The default implementation of the Web UI Framework. You can customize this implementation. Use the following interface classes:

  - YFSContext
  - YCPUIAPIManager

  The Web UI Framework provides the same transaction management functionalities as the previous version of the application.

Interface Contracts of the Web UI Framework - Transaction Management

For more information, refer to the Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

Interface Contract	Description	Methods
ISCUITransactionContext	ISCUITransactionContext defines the behavior expected in any implementation of transaction context in an application.	• begin Called on the beginning of the current transaction. Can be used to prepare connections to data sources. • commit Commits all of the changes for the current transaction. Called after the successful execution of all of the tasks for the current transaction. • rollback Called if any of the tasks for the current transaction fails. You can roll back all of the changes made during that transaction. • end Called when the current transaction ends. You can use this method to close all of the connections made to the data sources for that transaction. • addTransactionObject Adds multiple connections to a transaction context. • removeTransactionObject Removes connections to a transaction context. • getTransactionObject Fetches an already-added transaction object from the transaction context.
ISCUITransactionContextFactory	Defines the behavior expected in any implementation of a Transaction Context Factory in an application.	• createTransactionContext Creates a Transaction Context for a transaction. You can either create a new Transaction Context for fetch it from a Transaction Context pool and return it. • releaseTransactionContext Called when a transaction finishes. This method can either destroy a Transaction Context or return it to the pool. • init Instantiates a class when the first call for a Transaction Context is made. This method is called once for a Transaction Factory during instantiation. • sessionDestroyed

Interface Contract	Description	Methods
		Called when a session is destroyed. You can choose to perform an action based on the session destroyed. The ISCUITransactionContextFactory class extends the ISCUISessionAware class, which is a marker class that helps the ISCUITransactionContextFactory class register itself to the HttpSessionListener implementation class.

Transaction management also includes the following helper class which an application needs for transaction management-related tasks.

Class Name	Description	Methods
SCUITransactionContextHelper	Acts as a controller for transaction management.	• setTransactionContextFactory Registers the transaction context factory of an application. You can use this method to initialize and register a factory during the start of a web application. For any transaction context request, the SCUITransactionContextHelper class first looks to see if a factory is set. If not, the class tries to instantiate a factory using a context parameter. If a parameter is not found, an error is thrown. Once instantiated, the same instance is used for further references unless another context is set using the setTransactionContextFactory method. • getTransactionContextFactory Returns the Current Transaction Context Factory for an application. Returns **null** if no Factory is set or loaded.

Look and Feel in the Web UI Framework

UI Branding in the Web UI Framework

The Web UI Framework allows you to change the UI branding to your own brand name, including company logos. You can also change the theme and other items. A sample application can be divided into the following two parts:

- Themes
- Layout

Themes

All of the themes used in the Web UI Framework use CSS files that can be overridden by:

- Putting the overridden entries in a directory. A custom CSS file can be placed anywhere.

  Use the IncludeCSS tag to override the CSS file. The IncludeCSS tag supports a locale. Use the localized file of the CSS for the corresponding locale. The localized file can then be used to override the CSS.

  Example:

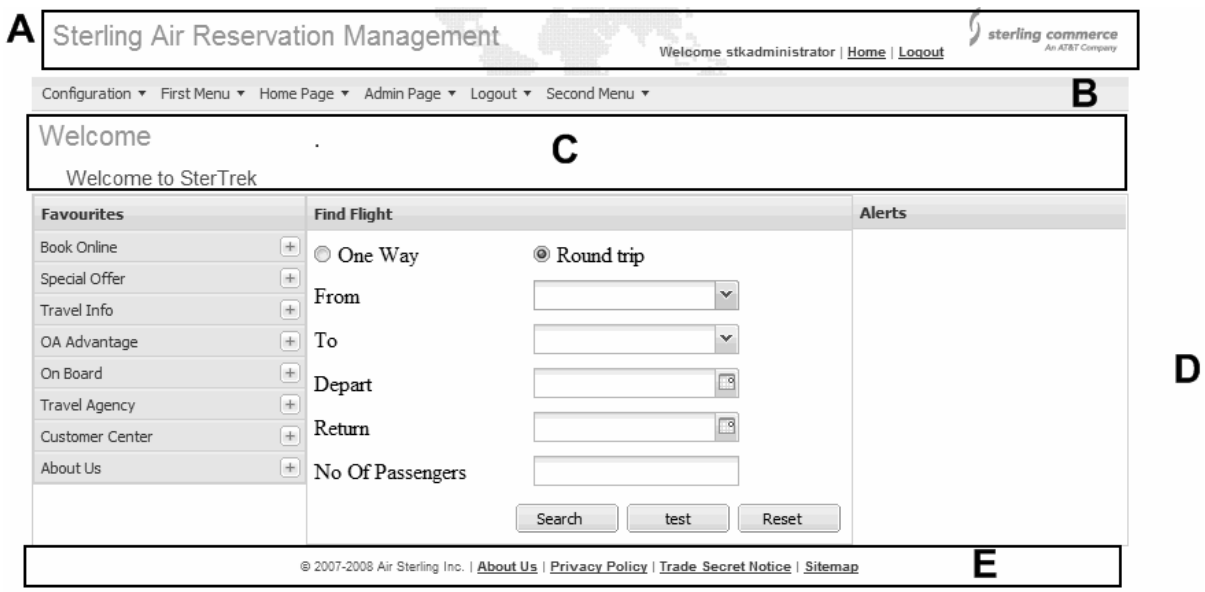
```
<scuitag:includeCSS path="/sfs/resources/default/css/sfs-core.css" />
```

- Using the post authentication implementation (ISCUIPostAuthenticationProvider).

Layout

This is a sample layout that can be used as a starting point by an application. You can choose to design your own custom layout but Sterling Commerce recommends that you follow this type of layout structure.

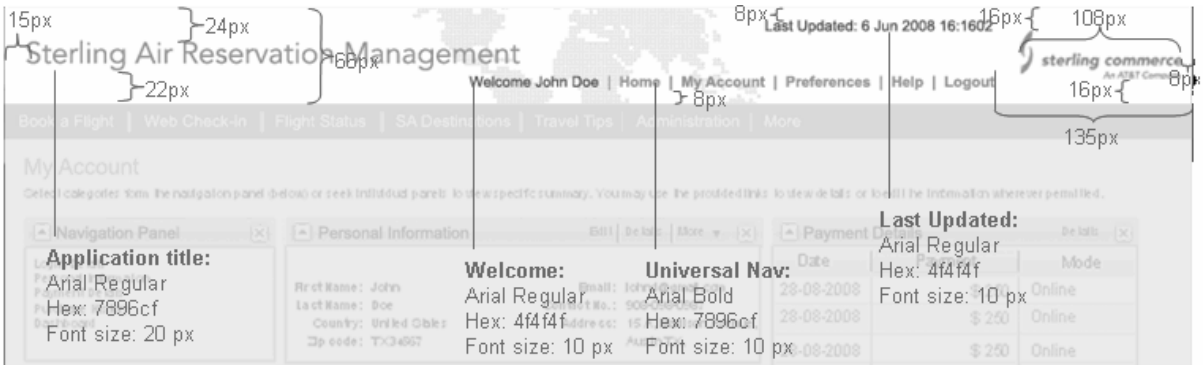
The screen layout can be divided into five different parts as shown below:



A—Header
B—Menu
C—Page Header
D—Page or Screen
E—Footer

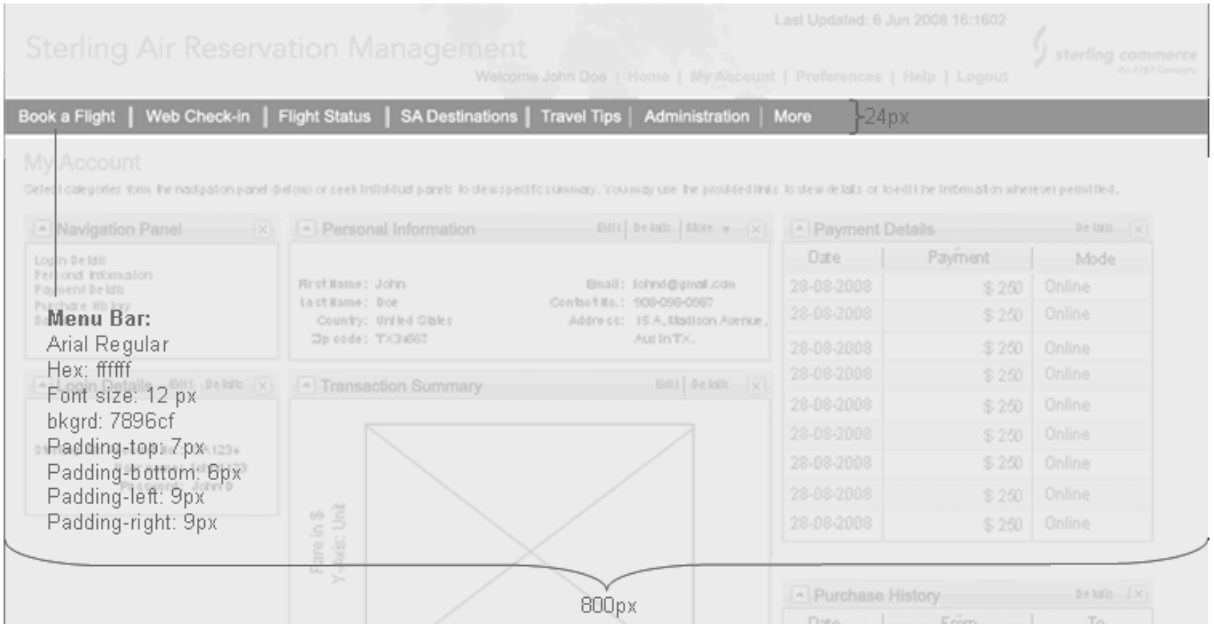
- Header—Consists of application name, company logo, static links, logged-in user's information, and a background image.

  Sample Header Layout with Spacing and Text Information:



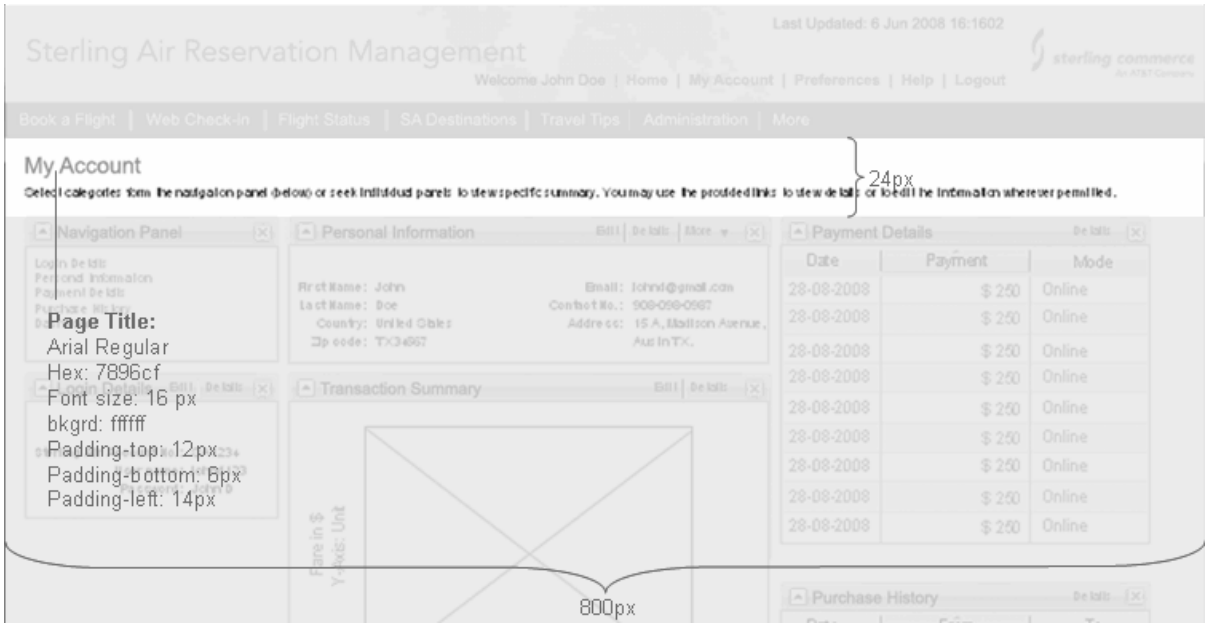
- Menu—The Web UI Framework provides the capability to include menu in a screen. Users can call a JavaScript function by passing the ID of the HTML element where the menu should be rendered. The menu entries are fetched from getUserHeirarchy API for the logged-in user.

  Sample Menu Layout with Spacing and Text Information:



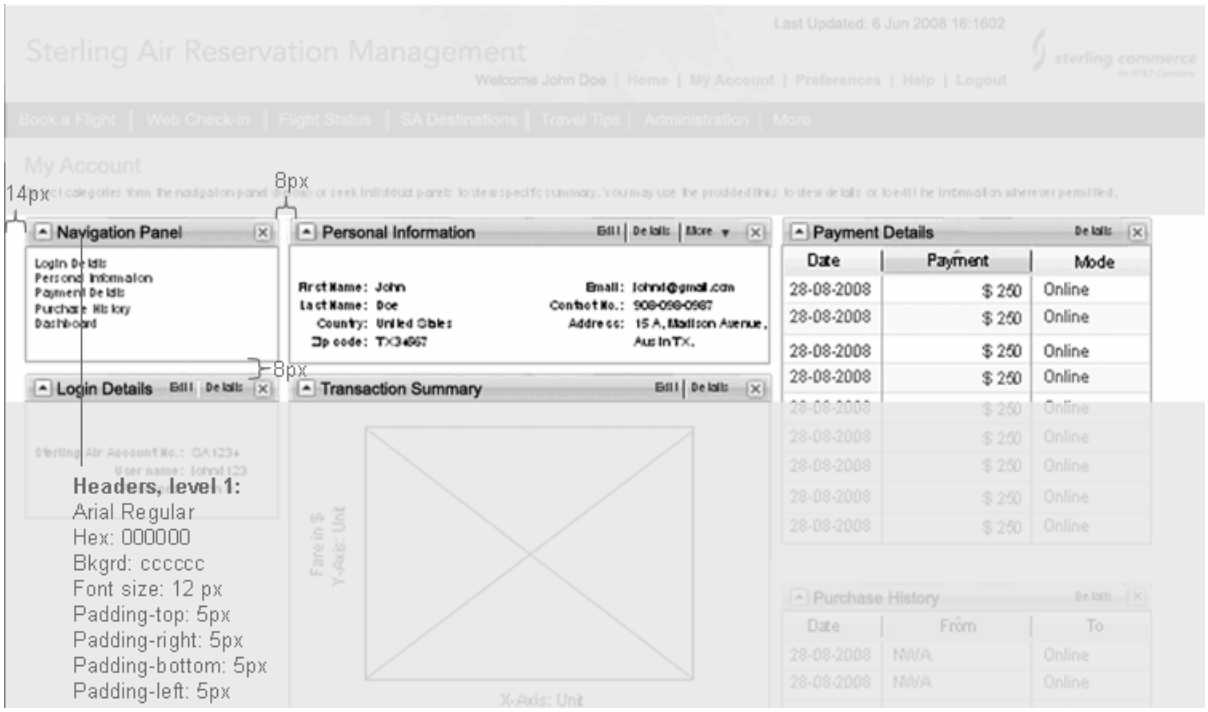
• Page Header—Consists of page title and one or more panels.

Sample Page Header Layout with Spacing and Text Information:



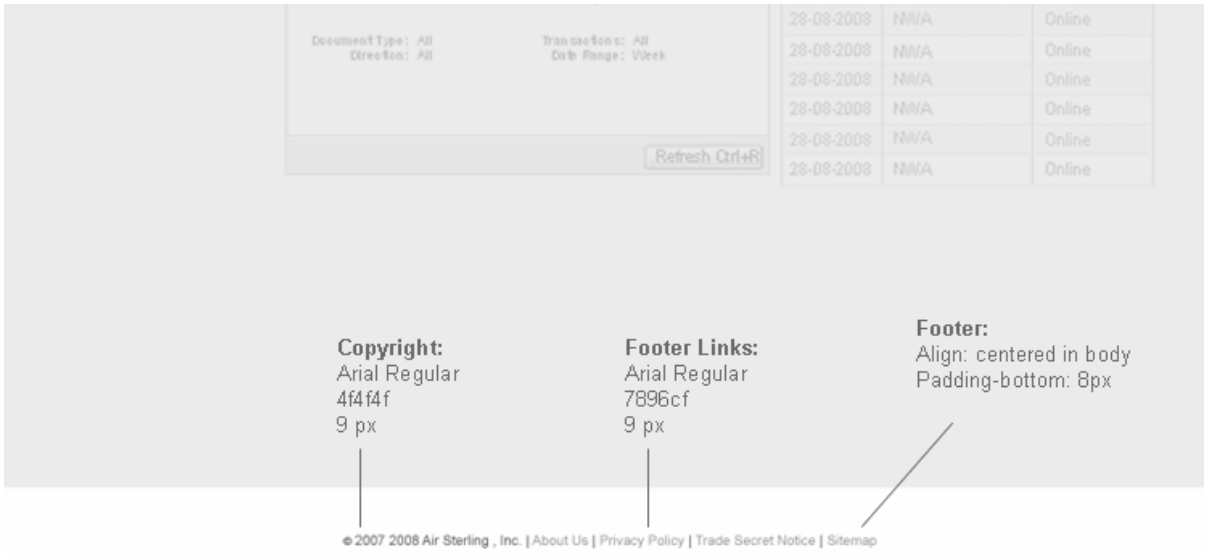
• Page or Screen—Consists of one or more panels.

Sample Page or Screen Layout with Spacing and Text Information:



• Footer—Consists of static links and copyright information.

Sample Footer Layout with Spacing and Text Information:



Specifying a Home Page when Building Screens with the Web UI Framework

During login, the authentication provider fetches the home page to be displayed in the following manner:

1. It first looks for the home page entry in the forward URL which is fetched from the request parameter's scui-login-page-referrer attribute.

   If the user is logging in for the first time, this attribute is set to **null** and the authentication provider looks for the custom Home Page Provider class entry. Only if the user is logging again after browsing through some pages, the authentication provider will first look for the home page to be displayed in the referral URL. For example, after logging in, the user has browsed through some pages and then for checkout he is again asked to re-enter the login information. Now, when the same user again tries to log in, the authentication provider will first look for the page to be displayed in the referral URL.

2. If the forwarded URL is not defined, it looks for the custom Home Page Provider class context param (scui-loginhomepage-provider) in the web.xml file.

   If you have some custom logic on how to display the home page based on some validations, then you will need to add this custom logic and validation in your Home Page Provider class. This class should implement the ISCUIHomePageProvider interface. You will return the URI for home page in the getHomePagePath(SCUIContext ctx) method.

   Using this custom home page provider, you can specify multiple home pages.

   You should add the context param entry for your custom Home Page Provider class in the web.xml file as shown below:

   ```
   <context-param>
         <param-name>scui-loginhomepage-provider</param-name>
         <param-value>com.sc.cp.MyHomePageProvider</param-value>
   </context-param>
   ```

3. If the Home Page Provider class entry is not defined, it looks for the Default Home Page context param (scui-loginhomepage-default) in the web.xml file.

   If you do not have any custom logic or validations for displaying the home page, you can provide a default home page to go to when the user is logged in. You should add a context param entry for the default home page in the web.xml file as shown below:

   ```
   <context-param>
         <param-name>scui-loginhomepage-default</param-name>
         <param-value>/<Web_Context_Root>/home.do</param-value>
   </context-param>
   ```

4. If the Default Home Page entry is not defined, it will go to the home.detail page.

Adding Keyboard Shortcuts with the Web UI Framework

The Web UI Framework allows to you add keyboard shortcuts on the UI components. For example, you can add a keyboard shortcut like **CTRL+ALT+s** for the **Search** button in the Search panel.

Trigger keys are case insensitive. Pressing **CTRl+ALT+S** is the same as pressing **CTRL+ALT+s**.

To add a keyboard shortcut, you need to add the triggers config option when you are defining the config for that particular Ext component. The triggers config parameter contains the following properties:

- triggerKey

  Sequence of keys that forms the shortcut key.

- triggerFn

  Actions that need to be performed when shortcut key is pressed on that component. For example, use doSearch when you want the shortcut key to trigger a search.

- triggerFnScope (Optional)

  Defines the scope in which the trigger function should be called. By default, the scope is the current component (this).

- triggerKeySeparator (Optional)

  separator key to be used for separating the sequence if keys defined for the shortcut key. By default, “+” is used as the separator key.

For example, if you want to enable the CTRL + ALT + S shortcut key for the **Search** button in the Search panel. The config for the Search panel will look like this:

```
items: [{
            xtype: "panel",
              . . .
              . . .
              . . .
            },
            buttons: [{
               . . .
               . . .
               . . .
                    handler: this.doSearch,
                    scope: this
             }]
            triggers: [{
                triggerkey: "Ctrl + Alt + s",
                triggerFn: this.doSearch,
                triggerFnScope: this,
                triggerKeySeparator: "+"
             }]
          }]
```

The triggers config parameter can contain an array of objects. For example, in the above example, if you want to define keyboard shortcut for the **Reset** button in the Search panel, the triggers config parameter will look like this:

```
triggers: [{
                triggerkey: "Ctrl + Alt + s",
                triggerFn: this.doSearch,
                triggerFnScope: this,
                triggerKeySeparator: "+"
             },
```

```
            {
               triggerkey: "Ctrl + Alt + r",
               triggerFn: this.doReset,
               triggerFnScope: this,
               triggerKeySeparator: "+"
        }]
```

Supporting Multiple Browsers with the Web UI Framework

All of the browsers that Ext JS JavaScript framework supports by default are supported in an application. You can add or delete certain browsers from this list.

Ext JS supports the following web browsers:

- Internet Explorer 6+
- FireFox 1.5+
- Opera 9+
- Safari 3+

Indicating Mandatory UI Fields with the Web UI Framework

The Web UI Framework provides a CSS class which allows an application to indicate a field as mandatory in the Web UI Framework screens. The mandatory fields in the UI will be indicated or marked using an asterisk symbol (*).

To indicate a field as mandatory, use the sc-mandatory class (defined in the *<Web_Context_Root>*\platform\css\platform.css file) on the label config to indicate that a field is mandatory.

Sample Ext JS config for a label:

```
{
   xtype: "label",
   sciId: "lblClosedOn",
   text: "Closed On",
   cls: "sc-mandatory"
}
```

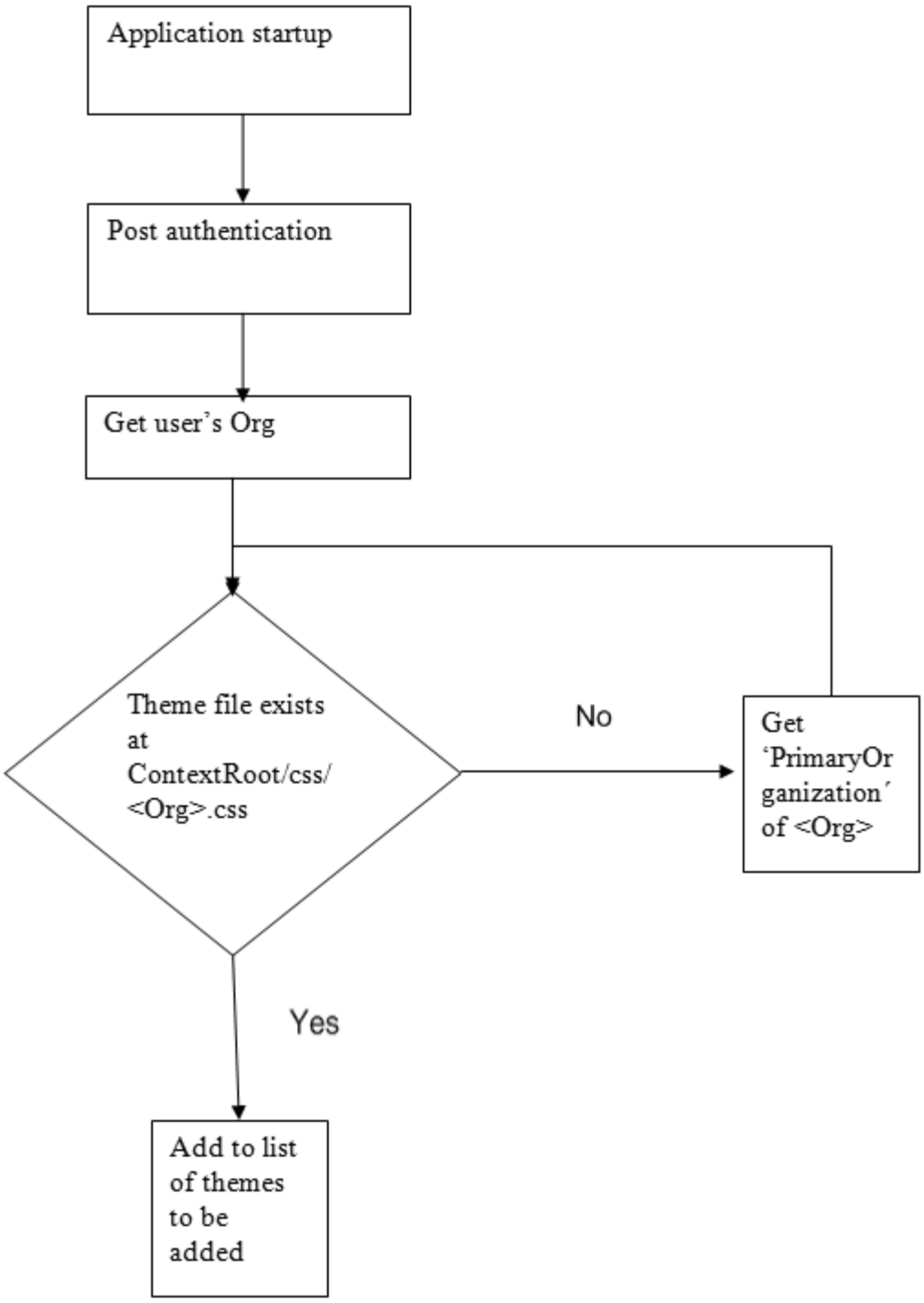
Adding Support for Custom Themes with the Web UI Framework

The Web UI Framework allows you to define your own custom themes. For example, you can have different themes based on organization. If the organization key (OrganizationKey) for user x is xyz, then you can define a new CSS file for this user as xyz.css. If that theme file exists, and the user is authenticated, the theme file will be loaded. If it does not exist, the application will look for xyz's primary organization (PrimaryOrganization) and load that organization's CSS file.

You can also add or modify these custom themes based on custom logic in the post authentication implementation.

For more information, refer to the Java API documentation for the ISCUIPostAuthenticationProvider interface in your installation directory.

The following chart shows how the theme file is determined:



To add support for custom theme files:

1. Set up your list of themes in an arrayList which will be added to the SCUITheme class. You can modify this arrayList if you have access to the SCUIContext class. The arrayList is created from the custom CSS file. For example:

```
private ArrayList customThemesList = new ArrayList();
public ArrayList getCustomThemesList() {
        return customThemesList;
}
public void addCustomThemes(String customTheme) {
        customThemesList.add(customTheme);
}
```

   Include a reference to your arrayList in an implementation of the ISCUIPostAuthenticationProvider interface. All of the custom themes can be added using the addCustomThemes method. For example:

```
public class SCUIPostAuthenticationProviderImpl implements
       ISCUIPostAuthenticationProvider {
     public SCUISecurityResponse postAuthenticate(SCUIContext uiContext) {
     ArrayList
list = uiContext.getUserPreferences().getTheme().addCustomThemes(<CUSTOM_THEME>);

     return new SCUISecurityResponse();
     }
}
```

   The custom CSS file name, along with the full path of the file, should be passed as the argument to the addCustomThemes method.

2. Implement the interface and the array list as a third party jar file to the application, using the install3rdparty.sh script.
3. Define the CSS by either creating a new CSS or by overriding the existing CSS.

About Dashboards, Dashlets, and Registries

A dashboard is a page that consists of multiple components, mostly independent of each other. Dashboards are reusable, customizable, can display different types of information at one place.

For example, dashboards can be used to present summarized views for a quick overview of various features. Or dashboards can be used as a directory of navigation and action links. Or dashboards can be used as a parallel processing engine where different components coexist and function independently.

Dashboard Components

A dashboard is composed of following two components:

Dashlet	A dashlet is an individual component that can be added to or removed from a dashboard. They are a reusable unit of functionality.
Dashboard	A dashboard is a container consisting of multiple dashlets. It controls the way a dashlet is organized in a dashboard.

Dashboards are controlled by metadata and are associated with two metadata registries:

Dashboard Registry	The dashboard registry contains the list of all available dashboards in an application. You must register the dashboard metadata definition with the dashboard registry before using it.
Dashlet Registry	The dashlet registry contains the list of all dashlet definitions in an application. You must register the dashlet metadata definition with the dashlet registry before adding the dashlet to the dashboard.

Creating and Registering Dashboard Metadata

The dashboard metadata definition is XML-based.

- To create dashboard metadata, you need to define the dashboard metadata definition XML.
- To register dashboard metadata, call the registerDashboard( ) method of the SCUIDashboardManager class (which is described in the *Platform Javadocs* resource).

Sample XML: Dashboard Metadata Definition

```
<Dashboards>
   <Dashboard id="myhomepage" title="My Homepage" tag="tag_1,tag_n"
              version="0.0.1">
      <Config>
         <Layouts  currentLayoutId="twoEqualColumn">
            <Layout id="twoEqualColumn" type="column"
                    previewIconUrl="/stk/img/dashboard/2col_preview.png">
               <Column id="d_one" width=".49"/>
               <Column id="d_two" width=".49"/>
            </Layout>
         </Layouts>
      </Config>
      <Dashlets>
         <Dashlet id="dashlet_one" parent="d_one" >
         </Dashlet>
         <Dashlet id="dashlet_two" parent="d_two">
            <Config numberOfColumns="1" >
            </Config>
         </Dashlet>
      </Dashlets>
   </Dashboard>
</Dashboards>
```

Attribute and element descriptions for the above sample are given below.

id	A unique identifier of the dashboard. This id used for checking resource permissions and authorization. If a dashboard contains a dashlet for which the user does not have the permission, that dashlet will not be shown to the user.
title	The title of the dashboard. It can be a bundle key which can be localized.
tag	A comma separated value of tag names of the dashboard. These tags are used to find dashlets having matching tags. These related dashlets are available when you customize the dashboard. Note: You will be able to see other tags also, but by default, only dashlets with matching tags will be displayed.

version	The dashboard metadata version. It can be used in case of multi-tenancy to support different versions of dashboard metadata.
Config	The Config element is used to define configuration options used to render the dashboard. One such configuration option is the layout of dashlets in a dashboard. Layouts are used to define layout strategies supported by a dashboard. It contains child elements defining individual layout strategies. Currently, only a column layout is supported as a standard layout of dashboard.
Dashlets	A list of all the dashlets to be displayed in the dashboard. Each dashlet can have following set of attribute and elements: • id. Reference id of the dashlet to be added to the dashboard. • Parent. The layout container id in which the dashlet will be added.

Creating and Registering Dashlet Metadata

The dashlet registry contains the list of all dashlet definitions in an application. You must register the dashlet metadata definition with the dashlet registry before adding the dashlet to the dashboard. Using the dashlet registry, you can first register your custom dashlets and then add those dashlets to a dashboard.

The dashlet metadata definition is XML-based.

- To create dashlet metadata, you define the dashlet metadata definition XML.
- To register dashlet metadata, call the registerDashlet( ) method of the SCUIDashboardManager class (which is described in the *Platform Javadocs* resource).

Sample XML: Dashlet Metadata Definition

```
<Dashlets>
   <Dashlet id="dashlet_one" title="Dashlet One" type="html" tag="tag_1">
      <Url>/stk/dashboard/dashlet_one.jsp</Url>
      <Imgurl>/stk/img/dashboard/dashlet_one_preview.png</Imgurl>
      <Description>
         Dashlet one description bundle key
      </Description>
      <Config numberOfColumns="3">
      </Config>
   </Dashlet>
   <Dashlet id="dashlet_two" title="Dashlet Two" type="extscreen" tag="tag_2"
            refreshable="Y">
      <Url>/stk/dashboard/dashlet_two.jsp</Url>
      <Imgurl>/stk/img/dashboard/dashlet_two_preview.png</Imgurl>
      <Description>
         Dashlet two description bundle key
      </Description>
      <Config numberOfColumns="2">
      </Config>
      <!-- Other metadata relevant to the dashlet can be added based on type
      and url ofthe dashlet -->
      <Group>
         <Section id="Airports" title="Airports" Icon="/stk/img/cog.png">
            <Link linkId="manage_airport" name="Manage Airports"
```

```
                  url="/stk/airport.do" />
         </Section>
         <Section id="Flights" title="Flight" icon="/stk/img/phone-icon.jpg">
            <Link linkid="manage_airport" name="Flights" url="/stk/flight.do" />

            <Link linkid="manage_airport" name="Flight Services"
                  url="/stk/flightService.do"/>
         </Section>
      </Group>
   </Dashlet>
   <Dashlet id="dashlet_three" title="Dashlet Three" type="html" tag="tag_3">
      <Url>/stk/dashboard/dashlet_three.jsp</Url>
      <Imgurl>/stk/img/dashboard/dashlet_three_preview.png</Imgurl>
      <Description>
         Dashlet three description bundle key
      </Description>
      <Config>
      </Config>
   </Dashlet>
</Dashlets>
```

Attribute and element descriptions for the above sample are given below.

id
A unique identifier of the dashlet. This id is used to query a dashlet and its properties. It is used by a dashboard definition to add a dashlet to it.

title
The title of the dashlet. It can be a bundle key which can be localized.

type
The type of the dashlet is a metadata about the type of content returned by a dashlet URL. By default, the Web UI Framework supports following types of dashlet:

- HTML. Response of type HTML must be a valid HTML document fragment. Following examples illustrate various ways in which it can be used to render the dashlet UI: (1) A fully rendered HTML fragment that can be viewed in a browser without any further processing. (2) An iframe that can be used to safely embed widget from external websites. (3) A scriptlet that can dynamically render the content in the container.
- extscreen. An extscreen is an ExtJS-based JavaScript UI rendering script. It can be used to render dynamic and interactive UI components.

tag
A comma separated value of tag names of the dashlet.

refreshable
Setting this value to Y makes the dashboard refreshable and a refresh icon is shown in the dashlet toolbox. When the refresh icon is clicked, the individual dashlet is refreshed.

URL
The URL used to get contents to render a dashlet. To render a dashlet into the dashboard, a request needs to be made to the server. The request is made to this URL and the response is rendered on the UI.

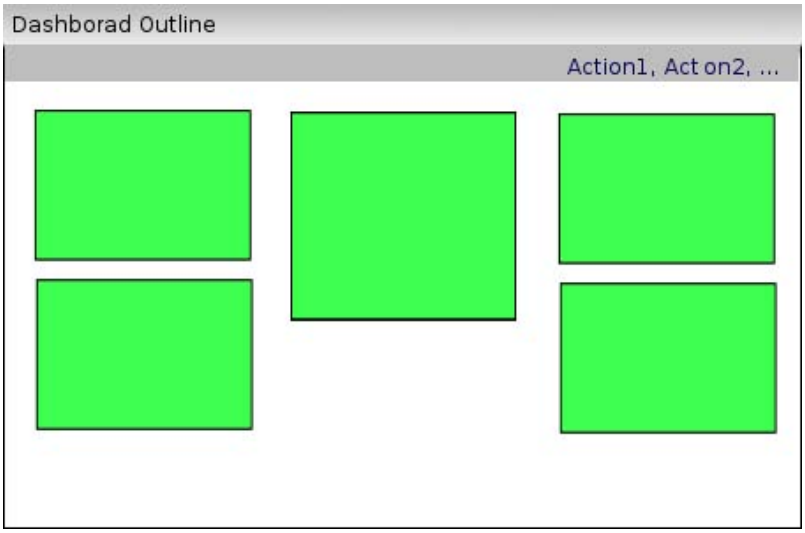
Imgurl
The URL to the image to be displayed showing a preview of the content displayed by the dashlet.

description
The description of the dashlet. It can be a bundle key which can be localized.

Config The Config element is used to define configuration options used to render the dashlet on the UI. Some dashlets that do not require a configuration option may choose not to specify the Config element. Note: In addition to the Config element, a dashlet can have other metadata elements. In general, the Config is used to define UI properties but it can be used to hold any kind of metadata.

About the Dashboard and Dashlet User Interface

The Web UI Framework provides a default implementation for the dashboard user interface. The UI screen is an ExtJS-based screen. It supports organizing dashlets in a column layout. The figure below shows the organization of a column layout based dashboard.



The dashboard UI consists of following components:

- Header or title
- Action bar for dashboard-related actions
- Content body where dashlets are organized in a column layout

The Web UI Framework provide a default dashlet container UI that can be used along with the dashboard UI. The figure below shows the sample dashlet container UI.

The dashlet container UI consists of following components:

- Header or title
- Toolbox in the header that contains three tools: close, refresh, and collapse

- Content body that is rendered using the dashlet metadata definition. The dashlet URL is used to generate the content body.

Creating the Dashboard User Interface

1. Create the JSON object for the dashboard and get the dashboard config.

   For example,

```
String dashboardId  = "MyDashboard";
JSONObject dashboardJSON;
String ctx = request.getContextPath( );
SCUIContext uiContext = SCUIContextHelper.getUIContext(request, response);
Element dashboardEl =
SCUIDashboardManager.getInstance( ).getUserDashboard(dashboardId, uiContext);
if (dashboardEl == null) {
   throw new SCUIException("invalid dashbardID :" + dashboardId);
}
Element dasboardConfigEl =
SCUIDashboardManager.getInstance( ).getDashboardConfig(dashboardEl, uiContext);
dashboardJSON = SCUIJSONUtils.getJSONObjectFromXML(dasboardConfigEl, uiContext);
dashboardJSON = dashboardJSON.getJSONObject(dasboardConfigEl.getNodeName( ));
```

2. Render the dashboard config on to the UI.

   For example,

```
<script type="text/javascript">
sc.plat.JSLibManager.loadLibrary("scuiPlatDashboard", function( ) {
Ext.onReady(function( ) {
var config = <%=dashboardJSON.toString()%>;
console.log('config is:', config);
var dashboard = new sc.plat.ui.Dashboard(config);
dashboard.render(Ext.getBody());
   });
});
</script>
```

Creating the Dashlet User Interface

Based on the dashlet type, you can create the dashlet user interface in two ways. For more information about these methods, refer to the *Platform Javadocs*.

Choose one method:

- If the dashlet is of type html, the output of the dashlet URL must be a standard HTML code fragment. For example,

```
<iframe height="260" frameborder="0" width=100% scrolling="no"
               src="../stk/modules/elements/news/iframe.html?format=300x250"

                 marginwidth="0"marginheight="0"/>
```

- If the dashlet is of type extscreen, the output of the dashlet URL must be a JavaScript code fragment, which should declare createBody( ) function. This function should create and return a valid Ext widget that will be rendered into dashlet body. For example,

```
function createBody( ) {
   return new Ext.Panel(...);
}
```

About Customizing and Resetting the Dashboard

Each dashboard user can customize the dashboard by adding a new dashlet or removing an existing dashlet. A dashboard user can also customize the layout or reposition the dashlets in the dashboard.

Whenever a dashboard user customizes the dashboard, a new metadata gets generated.The newly generated metadata is saved in each dashboard user's user preferences. Whenever a query is made by a client to the dashboard metadata for a dashboard user, the user-defined metadata is provided instead of the default definition. If a dashboard user did not customize the dashboard, then the default metadata definition is returned.

After customizing a dashboard, if a dashboard user wants to revert back to the default dashboard, a dashboard user can do that by using the resetting the dashboard option. When a dashboard user resets the dashboard, all the customizations that a dashboard user made to the default dashboard are removed and the dashboard user gets back the default dashboard.

Customizing the Dashboard User Interface with the Web UI Framework

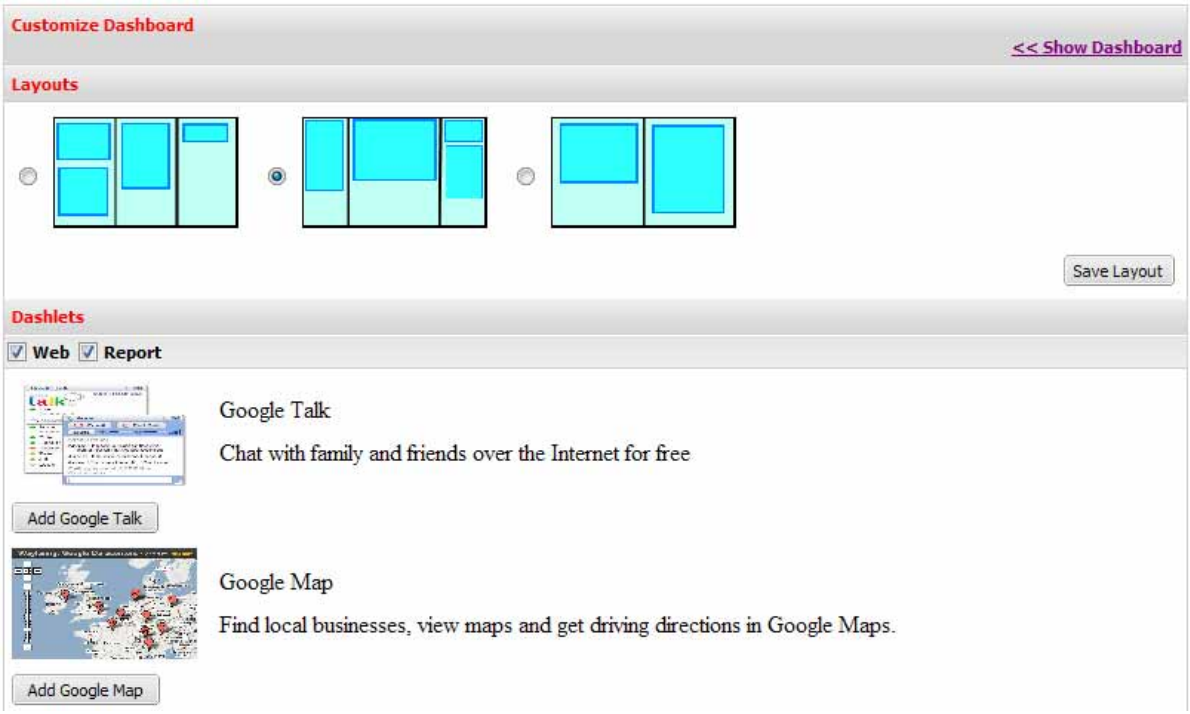
In the Web UI framework, you can customize the following attributes of the dashboard user interface:

- You can change the layout of the dashboard, including reorganizing and repositioning dashlets.
- You can add new dashlets and remove existing dashlets.

Each dashboard user has a personalized dashboard because all the customized dashboard metadata is saved for each user in his or her user preferences. Once the user has customized a dashboard, the default dashboard definition is not used.

1. In the Dashboard, click **Customize**.

   The Customize Dashboard window is displayed.



2. Make one of the following choices:
   - To change the layout, use the Layout panel and select one of the options. To view your changes, click **Show Dashboard**. To save your changes, click **Save Layout**.
   - To reposition dashlets, drag-and-drop the dashlet to the new location within the dashboard.
   - To remove a dashlet, select the one you want and click <X Insert Graphic>.
   - To add a dashlet, use the Dashlets panel and select the tag from which to add the dashlet. A list is displayed. Select the one you want to add and click **Save Layout**. To view your changes, click **Show Dashboard**.

Resetting the Dashboard to the Default Dashboard Definition

Users with customized dashboards can restore the default dashboard definition. This procedure must be completed if a new version of the application is installed that includes new dashboard definitions that users want to access.

1. For each dashboard user, display user preferences and remove all customized dashboard definitions.

   The default dashboard is displayed.
2. In the Dashboard, click **Reset Dashboard**.
3. In the Reset window, click **Yes**.

   The dashboard is reset to the default definition. After an application upgrade, the default dashboard includes any new definitions provided by the upgrade.

Extending an Existing Dashboard By Adding New Dashlets in the Web UI Framework

You can customize/extend an existing dashboard by adding dashlets. Below is sample XML, with dashlet metadata definitions:

```
<Dashlets>
   <Dashlet id="dashlet_one" title="Dashlet One" type="html" tag="tag_1">
      <Url>/stk/dashboard/dashlet_one.jsp</Url>
      <Imgurl>/stk/img/dashboard/dashlet_one_preview.png</Imgurl>
      <Description>
         Dashlet one description bundle key
      </Description>
      <Config numberOfColumns="3">
      </Config>
   </Dashlet>
   <Dashlet id="dashlet_two" title="Dashlet Two" type="extscreen" tag="tag_2"
            multiInstance="true" refreshable="Y">
      <Url>/stk/dashboard/dashlet_two.jsp</Url>
      <Imgurl>/stk/img/dashboard/dashlet_two_preview.png</Imgurl>
      <Description>
         Dashlet two description bundle key
      </Description>
      <Config numberOfColumns="2">
      </Config>
      <!-- Other metadata relevant to the dashlet can be added based on type
      and url of the dashlet -->
   </Dashlet>
   <Dashlet id="dashlet_three" title="Dashlet Three" type="html" tag="tag_3">
      <Url>/stk/dashboard/dashlet_three.jsp</Url>
      <Imgurl>/stk/img/dashboard/dashlet_three_preview.png</Imgurl>
      <Description>
         Dashlet three description bundle key
      </Description>
      <Config>
      </Config>
   </Dashlet>
</Dashlets>
```

Attribute and element descriptions for the above sample are given below.

id — A unique identifier of the dashlet. This id is used to query a dashlet and its properties. It is used by a dashboard definition to add a dashlet to it.

title — The title of the dashlet. It can be a bundle key which can be localized.

type

The type of the dashlet is a metadata about the type of content returned by a dashlet URL. By default, the Web UI Framework supports following types of dashlet:

- HTML. Response of type HTML must be a valid HTML document fragment. Following examples illustrate various ways in which it can be used to render the dashlet UI: (1) A fully rendered HTML fragment that can be viewed in a browser without any further processing. (2) An iframe that can be used to safely embed widget from external websites. (3) A scriptlet that can dynamically render the content in the container.
- extscreen. An extscreen is an ExtJS-based JavaScript UI rendering script. It can be used to render dynamic and interactive UI components.

tag

A comma separated value of tag names of the dashlet.

refreshable

Setting this value to Y makes the dashboard refreshable and a refresh icon is shown in the dashlet toolbox. When the refresh icon is clicked, the individual dashlet is refreshed.

multiInstance

Setting this to **true** would allow the dashlet to be added to the dashboard multiple times. This is an optional parameter and defaults to **false**.

URL

The URL used to get contents to render a dashlet. To render a dashlet into the dashboard, a request needs to be made to the server. The request is made to this URL and the response is rendered on the UI.

Imgurl

The URL to the image to be displayed showing a preview of the content displayed by the dashlet.

description

The description of the dashlet. It can be a bundle key which can be localized.

Config

The Config element is used to define configuration options used to render the dashlet on the UI. Some dashlets that do not require a configuration option may choose not to specify the Config element.

Note: In addition to the Config element, a dashlet can have other metadata elements. In general, the Config is used to define UI properties but it can be used to hold any kind of metadata.

To customize/extend an existing dashboard by adding dashlets, do the following:

1. Create the new XML file for the dashlets, using the above sample code as a guideline.
2. Do one of the following:
   - Navigate to the <*Install Dir*>/extensions/<*app-name*>/webpages folder. Create the desired directory structure and put your dashlet XML files here.

     The XML files defined here are copied to the <application-war>/extn/<your directory structure> folder.
   - Navigate to the <*Install Dir*>/repository/eardata/<*app-name*>/extn folder. Create the desired directory structure and put your dashlet XML files here.

     The XML files defined here are copied to the <application-war>/<your directory structure> folder.
3. Create a new servlet to load your XML files and add them to the user dashboard.

Sample code:

```
//<dirPath> = <your directory structure> or extn/<your directory
structure> depending on procedure used.
String appId = SCUIUtils.getApplicationId(config.getServletContext());
String dashletsFilePath = "<dirPath>/dashlets.xml";
InputStream isDlts = servletContext.getResourceAsStream(dashletsFilePath);
SCUIDashboardManager.getInstance(appId).registerDashlet(isDlts);
```

4. Package this servlet into a jar file and use the second method in case of a multi-war deployment. In other cases, you can use either method.
   - Run the Install3rdParty.sh script from <*Install Dir*>/bin to include this jar file in the application.
   - Navigate to <*Install Dir*>/repository/eardata/<*app-name*>/extn folder. Create the WEB-INF/lib folder. Copy the jar file here.
5. Navigate to the <*Install Dir*>/repository/eardata/<*app-name*>/extn folder. Make an entry in the web.xml file to load your servlet and ensure that it loads after your application initializer servlet by giving load-on-startup a value of around 1000 or greater.
6. Run the buildwar/buildear command.

   On launching the application, your new dashlets would be loaded and you would be able to view them when you select the customize link on the dashboard.

Note: Existing dashboard config and layouts cannot be changed during dashboard extensibility. Only new dashlets can be associated for a dashboard.

Note: To add a dashboard to an application, similar steps need to be followed. Below is sample XML, with dashboard metadata definitions:

```
<Dashboards>
   <Dashboard id="myhomepage" title="My Homepage" tag="tag_1,tag_n"
              version="0.0.1">
      <Config>
         <Layouts currentLayoutId="twoEqualColumn">
            <Layout id="twoEqualColumn" type="column"
                    previewIconUrl="/stk/img/dashboard/2col_preview.png">
               <Column id="d_one" width=".49"/>
               <Column id="d_two" width=".49"/>
            </Layout>
         </Layouts>
      </Config>
      <Dashlets>
         <Dashlet id="dashlet_one" parent="d_one" >
         </Dashlet>
         <Dashlet id="dashlet_two" parent="d_two">
            <Config numberOfColumns="1" >
            </Config>
         </Dashlet>
      </Dashlets>
   </Dashboard>
</Dashboards>
```

id A unique identifier of the dashboard. This id is used for checking resource permissions and authorization. If a dashboard contains a dashlet for which the user does not have the permission,that dashlet will not be shown to the user.

title The title of the dashboard. It can be a bundle key which can be localized.

tag A comma separated value of tag names of the dashboard. These tags are used to find dashlets having matching tags. These related dashlets are available when you customize the dashboard.

Note: You will be able to see other tags also, but by default, only dashlets with matching tags will be displayed.

---

version The dashboard metadata version. It can be used in case of multi-tenancy to support different versions of dashboard metadata.

Config The Config element is used to define configuration options used to render the dashboard. One such configuration option is the layout of dashlets in a dashboard.

Layouts are used to define layout strategies supported by a dashboard. It contains child elements defining individual layout strategies. Currently, only a column layout is supported as a standard layout of dashboard.

Dashlets A list of all the dashlets to be displayed in the dashboard. Each dashlet can have following set of attribute and elements:

- id. Reference id of the dashlet to be added to the dashboard.
- Parent. The layout container id in which the dashlet will be added.

About Chart Dashlets

Charts are a quick and easy way of presenting the summarized information. The Web UI Framework provides FusionCharts as an all purpose charting library. Chart dashlets can be used for presenting the summarized data (usually a list) to the user.

The purpose of a Chart dashlet is to provide a way to create a dashlet quickly from a given set of configuration options. Given the appropriate input, a Chart dashlet can be used to plot line charts, bar charts, and pie charts. This list is not inclusive and more types of charts can be created based on the input data source and configuration options.

The following input is required to create and render chart dashlet:

Input Data Source Input data source is used to provide data to chart. Input data source must be a mashup XML.

Chart Configuration The chart configuration is used to render the chart from the input data source. It contains attributes for extracting the relevant data from the input data source and plot it on the chart. Different types of charts will have different configuration options. Since Chart dashlets are based on fusion charts, the chart configuration options are mostly Fusion chart configuration options.

Creating Chart Dashlets

To create a chart dashlet, you need to define the chart dashlet metadata XML.

Sample XML: Chart Dashlet Metadata Definition

```
<Dashlet id="graphdashlet" title="My Graph " type="extscreen">
   <Url>/platform/dashboard/graphdashlet.jsp</Url>
   <Imgurl>/stk/img/dashboard/graphdashlet.jpg</Imgurl>
   <Description>View data in Graph</Description>
   <Config>
      <Chart chartType="MSCombi2D" showAboutMenuItem="1"
            rotateValues="1" caption="Monthly Flight expenses" xAxisName="Month"

             yAxisName="Expense" yAxisMinValue="10" yAxisValuesStep="2"
             yAxisMaxValue="2000" numberPrefix="Rs." enableSmartLabels="1"
             enableRotation="1" showValues="0" useRoundEdges="1"
             formatNumberScale="1" showBorder="1" chartOrder="line,area,column"
             endAngX="35" endAngY="-20">
      </Chart>
      <Datasources>
         <Datasource mashupId="flightBookingGraphMashup" seriesName="expence"
                     renderAs='Column'>
         </Datasource>
      </Datasources>
      <categories>
         <category label='JAN' />
         <category label='FEB' />
         <category label='MAR' />
         <category label='APR' />
         <category label='MAY' />
         <category label='JUN' />
         <category label='JUL' />
         <category label='AUG' />
         <category label='SEP' />
         <category label='OCT' />
         <category label='NOV' />
         <category label='DEC' />
      </categories>
   </Config>
</Dashlet>
```

Attribute and element descriptions for the above sample are given below.

type Specifies the type of screen being used for displaying the chart dashlet UI. By default, the chart dashlet provided by platform will be of type extscreen.

URL The URL used to get contents to render the chart dashlet. Note: The Url for chart dashlet should be /platform/dashboard/graphdashlet.jsp

Chart The attributes of this element are valid FusionCharts chart attributes. For various chart specific attributes, refer to the FusionChart documentation.

Datasources/Datasource A datasource is used to provide the dashlet with the relevant data to display as chart. One datasource corresponds to one series to be drawn in the chart. Using multiple datasources multiple series can be displayed in a chart. Moreover, a datasource definition has metadata to call a mashup. The input to the mashup is the datasource XML element and the output is expected to be an XML element. A datasource element has following attributes:

- mashupId. The id of the mashup to be called.
- seriesName. Used to define the series name.
- renderAs. Used to define the series type.

Note: The Datasource output should be a valid dataset in the format expected by FusionCharts. For example:

```
<dataset>
<set label="label_1" value="001" />
<set label="label_2" value="002" />
</dataset>
```

categories Specify the categories as required by FusionChart to plot data on the axes.

Security with the Web UI Framework

Web UI Framework Security - Authentication

Authentication identifies users who have access to the application. It is the first step in the login process. It occurs before you are authorized for resources in the application. Use the Application Manager to specify user IDs and passwords.

All requests are authenticated unless the URI (universal resource indicator) is in the bypass list. This is sometimes done for graphic files, cascading style sheets (css), and other items that support information that is already protected by authentication.

With the Web UI Framework, you have the following options for implementing authentication:

- The default implementation, which includes support for single sign on (SSO).

  If you are currently using the default implementation of authentication, and want to continue using that implementation, you must use this option. The default implementation supports all existing authentication features.

- A custom implementation where you plug in your own authentication implementation and do not use the default implementation. A customized implementation can have additional authentication processes, such as single sign on (SSO). You also can customize the post authentication mechanism.

  You must use either the default authentication implementation or a customized authentication implementation, but if you do not use the default post authentication implementation, you are not required to provide a customized post authentication implementation.

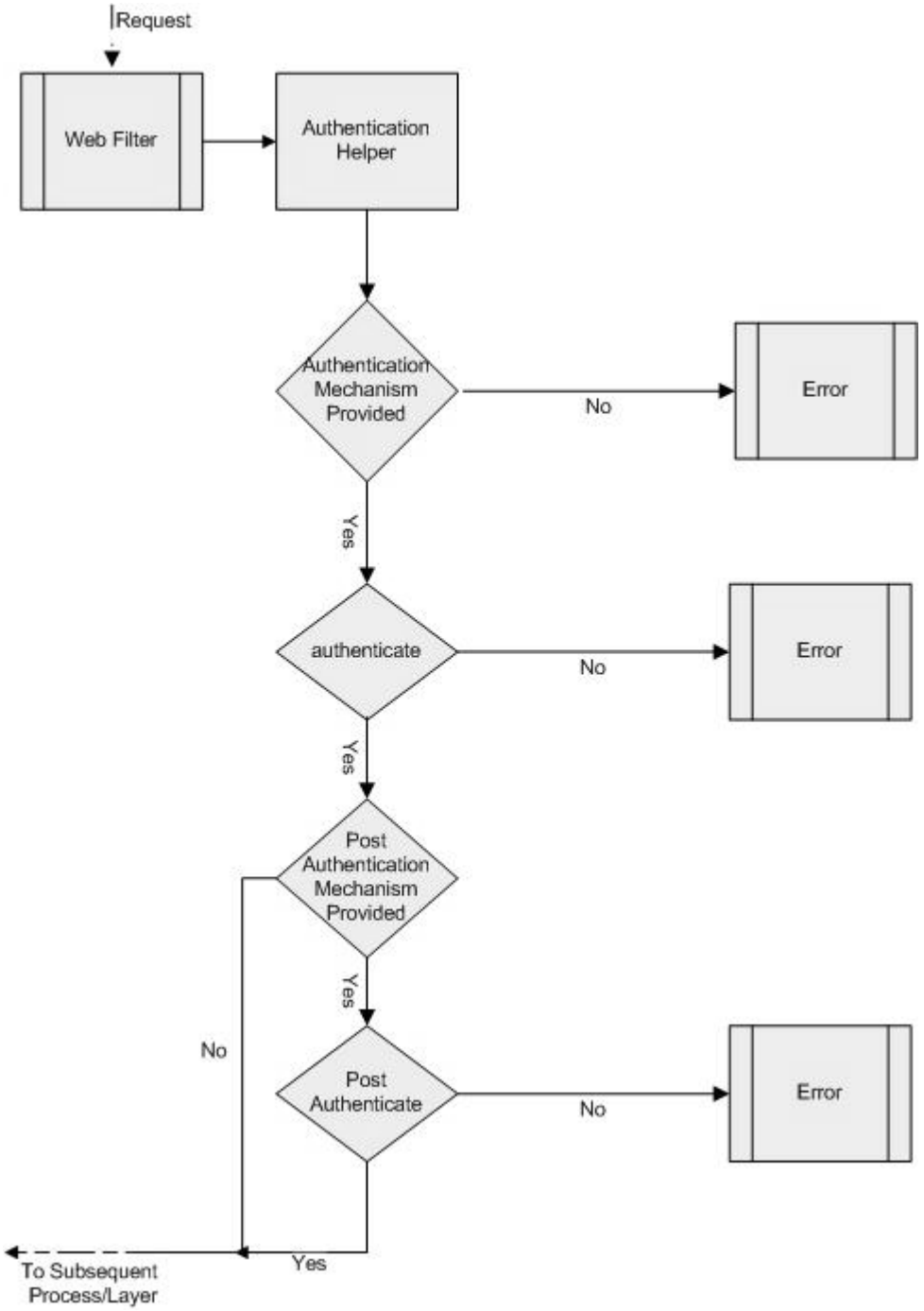
- A custom implementation where you customize the default implementation.

With all options, the implementation is plugged into interface contracts, which have definitions of the behavior expected with any authentication mechanism that can plug in to it. This ensures a consistent mechanism for authentication, no matter how you are implementing it (custom or default). The interface contracts also have definitions of the behavior expected with any post-authentication mechanism, which is called if the authentication mechanism succeeds.

Authentication can be invoked in different ways:

- LDAP
- Database table

The following picture shows the flow of authentication:



Web UI Framework Security - Implementing Authentication

When you implement authentication, you must first decide if you want to customize or use the default implementation of authentication provided by the application. You have the following options:

- The default implementation.

  To use this implementation, just install the application.
- A customized implementation where you plug in your own authentication implementation and do not use the default implementation.
- A customized implementation where you customize the default implementation.

Customizing Authentication

The custom authentication mechanism for the application consists of the AuthenticationProvider class that implements the ISCUIAuthenticationProvider interface. AuthenticationProvider is plugged in using the context parameter in web.xml as shown in the following example:

```
<context-param>
   <param-name>scui-authentication-provider</param-name>
   <param-value>com.app.MyAppAuthenticationProvider</param-value>
 </context-param>
```

The following shows an example of a custom AuthenticationProvider that uses the provider specified in the web.xml example:

```
public class MyAppAuthenticationProvider
implements ISCUIAuthenticationProvider
 {
  public SCUISecurityResponse authenticate(SCUIContext uiContext)
  {
    //authenticate the user
    //set the SCUISecurityContext in uiContext
    //set the SCUIUserPreferences in uiContext
    ....
  }

  public void init()
  {
    //initialize the authentication mechanism.
    ...
  }

  public void sessionDestroyed(HttpSessionEvent sessionEvent)
  {
    //close the connection and release it back into the pool
    ...
  }
 }
```

Interface Contracts of the Web UI Framework - Authentication

For more information, refer to the Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

Interface Contract	Description	Methods
ISCUIAuthenticationProvider	Defines the behavior expected in any implementation of authentication in an application. AuthenticationProvider is plugged in to an application using the context parameter in web.xml: • <param-name> scui-authentication-provider • <param-value> com.app.MyAppAuthenticationProvider The ISCUIAuthenticationProvider class extends the ISCUISessionAware class, which is a marker class that helps the ISCUIAuthenticationProvider class register itself to the HttpSessionListener implementation class.	• authenticate Takes in the SCUIContext. The expected response is in the form of the SCUISecurityResponse object that encapsulates the return status, the URL of the page, exception, and error message. The AuthenticationProvider class needs to set SCUISecurityContext and SCUIUserPreferences in SCUIContext if the user is authenticated. • init Handles initialization, like loading the security information or caching it. This method is called once, when AuthenticationProvider is first set. • sessionDestroyed Closes all opened session-specific handles.

Interface Contracts of the Web UI Framework - Post Authentication

For more information, refer to the Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

Interface Contract	Description	Methods
ISCUIPostAuthenticationProvider	Defines the behavior expected in any implementation of authentication in an application. AuthenticationProvider is plugged in to an application using the context parameter in the web.xml file: • <param-name> scui-post-authentication-provider • <param-value> com.app.MyAppPostAuthenticationProvider Multiple PostAuthenticationProviders can be set using the web.xml file. No order is maintained but if one PostAuthentication fails, the request is redirected to the URL in the forwarded page with the error. Multiple PostAuthenticationProviders are set using the web.xml param-name **scui-post-authentication-provider**.	• postAuthenticate Takes in the SCUIContext. The expected response (after post authentication) is an SCUISecurityResponse object that encapsulates the return status, the URL of the page, exception, and error message. • init Handles initialization, like loading the security information or caching it. This method is called once, when PostAuthenticationProvider is first set. • sessionDestroyed Closes all opened session-specific handles. The ISCUIAuthenticationProvider class extends the ISCUISessionAware class, which is a marker class that helps the ISCUIAuthenticationProvider class register itself to the HttpSessionListener implementation class.

Interface Contract	Description	Methods
		When a session is invalidated or destroyed, the sessionDestroyedmethod is called by the listener to close the handles opened during initialization.

Web UI Framework Security - Bypassing Authentication for a URI

You can set up the application to bypass authentication for URIs (universal resource indicators) that point to graphic files, cascading style sheets (css), and other items that support information that is already protected by authentication.

1. Open the web.xml file.
2. Add one or more parameters to the <context-param> tag:

   • bypass.uri.endswith

   Allows any URI ending with the specified text to be bypassed. You can use this parameter with js, css, and gif files.

   In the following example, any URI that ends with ".gif" would be bypassed:

   ```
   <context-param>
     <param-name>bypass.uri.endswith.<application>.3</param-name>
     <param-value>.gif</param-value>
   </context-param>
   ```

   • bypass.uri.regex

   Allows any URI that includes the specified wild card characters to be bypassed.

   In the following example, any URI that includes "*<app_dir>*", "/", and at least one uppercase letter would be bypassed:

   ```
   <context-param>
     <param-name>bypass.uri.regex.<application>.1</param-name>
     <param-value>.*<app_dir>/[A-Z]+.*</param-value>
   </context-param>
   ```

Web UI Framework Security - Authorization

Authorization enables you to grant permissions to a user for different resources. It occurs after you are authenticated in an application.

With the Web UI Framework, you have the following options for implementing authorization:

• The default implementation.

If you are currently using the default implementation of authorization, and want to continue using that implementation, you must use this option. The default implementation supports all existing authorization features.

• A customized implementation without the default implementation.
• A customized implementation of the default implementation.

With all options, the implementation is plugged into interface contracts, which have definitions of the behavior expected with any authorization mechanism that can plug into it. This ensures a consistent mechanism for authorization, no matter how you are implementing it (custom or default).

If you do not use the Web UI Framework default implementation of authorization, and no custom implementation is provided, by default the user will have access to all resources.

Authorization uses a resource ID to see if a user has permission to use a resource. Resource IDs control access to the Extensibility Workbench and Designer Workbench.

Authorization can be invoked in different ways:

• LDAP
• Database table
• A resource ID in the metadata of a mashup

  To use a mashup, you must define a resource ID for the mashup to control the access of the mashup and give it to the mashup.xml.

Web UI Framework Security - Implementing Authorization

When you implement authorization, you must first decide if you want to customize or use the default implementation of authorization provided by the application. You have the following options:

• The default implementation.

  To use this implementation, just install the application.

• A customized implementation without the default implementation.
• A customized implementation of the default implementation.

Sterling Commerce recommends that permissions for users be cached.

Customizing Authorization

The custom authorization mechanism for the application consists of the AuthorizationProvider class that implements the ISCUIAuthorizationProvider interface and ResourcePermission that implements the ISCUIResourcePermission interface. ResourcePermission is returned by the AuthorizationProvider class after the authorization. AuthorizationProvider is plugged in using the context parameter in web.xml as shown in the following example:

```
<context-param>
   <param-name>scui-authorization-provider</param-name>
   <param-value>com.app.MyAppAuthorizationProvider</param-value>
</context-param>
```

You can generate resource permission code using the resource permission template of the Code Template Generator.

The following shows an example of a custom AuthorizationProvider that uses the provider specified in the web.xml example:

```
public class MyAppAuthorizationProvider implements
ISCUIAuthorizationProvider
```

```
{
   ....
   public boolean hasPermission(SCUIContext uiContext, String resourceId)
      {
        ISCUIResourcePermission getPermission(uiContext, resourceId);
        ....
        }
   public ISCUIResourcePermission getPermission(SCUIContext uiContext,
String resourceId)
     {
       //authorize the user from the SCUISecurityContext
       ...
     }
    public void init()
     {
       // initialize the authorization mechanism.
       ...
      }
    public void sessionDestroyed(HttpSessionEvent sessionEvent)
    {
       //close the connection and release it back into the pool      ...
    }
}
```

Interface Contracts of the Web UI Framework - Authorization

For more information, refer to the Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

Interface Contract	Description	Methods
ISCUIAuthorizationProvider	Defines the behavior expected in any implementation of authorization in an application. ISCUIAuthorizationProvider is plugged in to an application using the context parameter in web.xml: • <param-name> scui-authorization-provider • <param-value> com.app.MyAppAuthorizationProvider	• hasPermission Takes in SCUIContext and resourceId. Returns true if the user in the SecurityContext has any permission to the resource given by the resourceId. Otherwise, it returns false. • getPermission Takes in SCUIContext and resourceId. Returns an implementation of ISCUIResourcePermission that contains the permission for the given resourceId. • init Handles initialization, like loading the security information or caching it. This method is called once, when ISCUIAuthorizationProvider is first set. • sessionDestroyed Closes all opened session-specific handles.

Interface Contract	Description	Methods
		The ISCUIAuthorizationProvider class extends the ISCUISessionAware class, which is a marker class that helps the ISCUIAuthorizationProvider class register itself to the HttpSessionListener implementation class. When a session is invalidated or destroyed, the sessionDestroyedmethod is called by the listener to close the handles opened during initialization.
ISCUIResourcePermission	Defines the behavior expected in any implementation of authorization for a given resource ID in an application. ISCUIResourcePermission is returned by ISCUIAuthorizationProvider after the authorization.	• canRead Returns true if the user has permission to read for a given ResourceId. Otherwise, it returns false. • canEdit Returns true if the user has permission to edit for a given ResourceId. Otherwise, it returns false. • canExecute Returns true if the user has permission to execute for a given ResourceId. Otherwise, it returns false. This could be the permission control that is used for executing the mashup class.

Web UI Framework Security - Adding Login Pages

The Web UI Framework enables you to set up more than one login page. Login pages can be used for different organizations or other groupings of users. You also can set up a customized implementation of multiple login pages.

1. Install the application with the Web UI Framework.
2. Decide how you want to implement multiple login pages, using one of the following ways:
   - The default implementation provided in the Web UI Framework.

     If no customized login page provider is given, the default implementation reads the following web.xml file `<context-param>` parameters:

     ```
     <param-name>scui-login-page</param-name>
     <param-value>/myapp/console/login.jsp</param-value>
     ```

   - A customized implementation that is plugged into the Web UI Framework.

3. For a customized implementation, specify your custom login page provider in web.xml using the scui-login-page-provider parameter.

   Example:

   ```
   <context-param>
     <param-name>scui-login-page-provider</param-name>
     <param-value>com.app.MyLoginPageProvider</param-value>
   </context-param>
   ```

   This provider will be accessed by the getLoginPage() method of ISCUILoginPageProvider. The custom implementation must use the interface contract defined in the ISCUILoginPageProvider class.

   To implement the customized Java code, build a jar file that contains the Java class, and then install the jar file using the install3rdparty.sh script. To implement this customization, rebuild the EAR or WAR file as you did during the installation, and then deploy the application on the application server.

Web UI Framework Security - Supporting Multiple Guest Users

With the Web UI Framework, your authentication process can include the authentication of one or more guest users for a particular URL of the application. If the application is not configured for multiple guest users, the default implementation allows only one guest user (if it is set up). If the application is not configured for multiple guest users or for the default implementation of only one guest user, the guest can be specified using web.xml parameters:

```
<context-param>
   <param-name>scui-guest-user</param-name>
   <param-value>myAppGuest</param-value>
</context-param>
```

To support multiple guest users, you must use the default implementation of the Web UI Framework. You can customize this default implementation.

1. Install the application with the default implementation of the Web UI Framework.
2. Specify your custom guest user provider in web.xml using the scui-guest-user-provider parameter.

   Example:

   ```
   <context-param>
       <param-name>scui-guest-user-provider</param-name>
       <param-value>com.app.MyGuestUserProvider</param-value>
   </context-param>
   ```

   This provider will be accessed by the getGuestUser() method of ISCUIGuestUserProvider. For all guest users, the password is the same as the user name. The custom implementation must use the interface contract defined in ISCUIGuestUserProvider.

   To implement the customized Java code, build a jar file that contains the Java class, and then install the jar file using the install3rdparty.sh script. To implement this customization, rebuild the EAR or WAR file as you did during the installation, and then deploy the application on the application server.

Web UI Framework Security - Adding Request Validators

The Web UI Framework allows you to set up more than one validation for a request. This validation process requires additional authentication of a user after that user has initially logged in. It allows that user to continue a login session.

For more information, refer to the Java API documentation in your installation directory (<*INSTALL_DIR*>/xapidocs/core_javadocs).

1. Install the application with the default implementation of the Web UI Framework.
2. Create your implementation of multiple validations, which you will plug into the Web UI Framework. The Web UI Framework does not have a default implementation of multiple validations. If no implementation is provided, the request is not further validated after the initial authentication.

   The request validations are done for every request, so you need to optimize this feature based on your needs. The implementation of request validators must use the contract defined in ISCUIRequestValidator.
3. The SCUISecurityResponse class is returned by the request validator's validate method. If the validation fails, the request is redirected to the URL specified in the SCUISecurityResponse class. Also, include settings for the return status, exception, and error message. This information is used by the validate method of the ISCUIRequestValidator in the Web UI Framework.

   The ISCUIRequestValidator interface defines what the Web UI Framework expects in any request validation implementation. This interface uses the following methods:

   • validate

   Takes in SCUIContext. The response is an SCUISecurityResponse object that encapsulates the return status, the URL of the page, exception, and error message. This method executes the business logic needed by the application.

   • init

   Handles initialization.

   • sessionDestroyed

   Closes all opened session-specific handles. The ISCUIValidator extends the ISCUISessionAware interface, a marker interface that will facilitate ISCUIValidator to register itself to the HttpSessionListener implementation class. When the session is invalidated or destroyed, the sessionDestroyed method is called by the listener to close the session-specific handles opened during initialization.

   The following shows an example of an ISCUIRequestValidator interface:

```
public interface ISCUIRequestValidator extends ISCUISessionAware
{
  public SCUISecurityResponse validate(SCUIContext uiContext);
  public void init();
  public void sessionDestroyed();
}
```

   The request validation consists of one or more instances of RequestValidator that implements the ISCUIRequestValidator interface class. Multiple request validators can be set, but their order is not

guaranteed. RequestValidator is plugged in using the context parameter in web.xml as shown in the following example:

```
<context-param>
   <param-name>scui-request-validator1</param-name>
   <param-value>com.app.MyURLValidator</param-value>
</context-param>
<context-param>
   <param-name>scui-request-validator2</param-name>
   <param-value>com.app.MyAdminValidator</param-value>
</context-param>
```

All of the validation implementation or validators given in the context parameter in web.xml are called (in no particular order) for supporting additional validation.

4. To implement the customized Java code, build a jar file that contains the Java class, and then install the jar file using the install3rdparty.sh script. To implement this customization, rebuild the EAR or WAR file as you did during the installation, and then deploy the application on the application server.

Web UI Framework Security - Cross-Site Request Forgery

The Web UI Framework provides protection for the application against cross-site request forgery (CSRF), which maliciously exploits a web site where unauthorized commands are transmitted from a user that the web site trusts. CSRF (also called XSRF) is different from cross-site scripting (CSS or XSS), which exploits the trust a user has for a particular site. CSRF is also known as one-click attack, sidejacking, or session riding.

CSRF works by including a link or script in a page that accesses a site to which the user is known (or is supposed) to have authenticated. For example, User A might be browsing a forum where User B has posted a message. With CSRF, User B might create the following HTML image element that, instead of being an image file, references a script on the web site of User A's bank and requests a withdrawal of $1,000,000:

```
<img src="http://bank.example/withdraw?amount=1000000&for=USER-B">
```

If User A's bank keeps their authentication information in a cookie, and if the cookie hasn't expired, then the attempt by User A's browser to load the image will submit the withdrawal form with the authentication cookie, and authorize a transaction without User A's approval.

In this scenario, the problem can be summed up in the following three points:

- Because of the browser's policy, the authentication cookies are sent to the bank server even though the request originated from a different web site.
- User A's bank stores authentication information in a cookie and completely relies on the cookies for authentication purposes.
- User A's bank does not differentiate between GET and POST requests.

The CSRF protection in the Web UI Framework does not apply to the first point, since it is a browser policy. But it does apply to the second and third points by using both a cookie and an additional token for authentication. CSRF attacks are usually prevented by always checking for a unique token in each request that hits the server. In the Web UI Framework, the token is used in the following manner:

1. When login finishes, a newly created token is set for the session (for validation purposes). This token is available on the client side of the application.
2. The token is used in the following ways:

- This token is used for all AJAX requests and within the Web UI Framework utilities.
- When a POST or GET request is made to the server, the application automatically validates that the CSRF token is available in the request.

Web UI Framework Security - Protecting Against CSRF Attacks

1. Open the web.xml file.
2. To validate the token that is used to protect against CSRF attacks, create a request validator that will be registered in the application (if the validator is not already present in the web.xml file).

   Example:

   ```
   <context-param>
     <param-name>scui-request-validator-10</param-name>
     <param-value>
   com.sterlingcommerce.ui.web.platform.security.SCUICSRFTokenValidator
     </param-value>
   </context-param>
   ```

3. Set up the modes in which the validator can operate:

   - ALL (default) - Both POST and GET requests will be validated for the CSRF token.
   - POST - Only POST requests will be validated for the CSRF token.
   - NONE - The validator will not validate any request for the CSRF token.

   You can specify the validator mode in the context parameter of either the config.xml file or the web.xml file (if the validator mode is not already present in the web.xml file).

   The mode defaults to **ALL** if the mode is not specified or if a context parameter is not specified for the validate mode.

   Example:

   ```
   <context-param>
      <param-name>scui-csrf-validator-request-method</param-name>
      <param-value>ALL</param-value>
   </context-param>
   ```

4. If necessary, set up URI inclusion and exclusion lists for the validator, using the following guidelines:

   - If a URI is on the exclusion list, it will not be validated for the CSRF token.
   - If a URI (universal resource indicator) is on the inclusion list, and not on the exclusion list, it will be validated for the CSRF token.
   - If a URI is not on the exclusion list and is in the inclusion list, it will be validated for the CSRF token.

   Use the following context parameters in the web.xml file to create inclusion and exclusion lists. Any number of parameters can be provided.

   - csrf-include-uri

     Any request with a URI that is the same as the value is validated for the CSRF token.

Example (for web.xml):

```
<context-param>
   <param-name>csrf.include.uri.endswith.stk.1</param-name>
   <param-value>.do</param-value>
</context-param>
```

- csrf-include-uri-endswith

  Any request with a URI that ends with the value is validated for the CSRF token.

  Example (for web.xml):

```
<context-param>
  <param-name>csrf.include.uri.endswith.stk.2</param-name>
  <param-value>.xml</param-value>
</context-param>
```

- csrf-include-uri-regex

  Any request with a URI that matches the regex (provided as a value for the parameter) is validated for the CSRF token.

  Example (for web.xml):

```
<context-param>
  <param-name>csrf.include.uri.stk.1</param-name>
  <param-value>/stk/home.jsp</param-value>
</context-param>
```

- csrf-bypass-uri

  Any request with a URI that matches the value is bypassed and not checked for the CSRF token.

  Example (for web.xml):

```
<context-param>
  <param-name>csrf.bypass.uri.stk.1</param-name>
  <param-value>/console/login.jsp</param-value>
</context-param>
```

- csrf-bypass-uri-endswith

  Any request with a URI that ends with the value is bypassed

  Example (for web.xml):

```
<context-param>
   <param-name>csrf.bypass.uri.endswith.stk.1</param-name>
   <param-value>.js</param-value>
</context-param>
```

- csrf-bypass-uri-regex

  Any request with a URI that matches the regex (provided as a value for the parameter) is not checked for the CSRF token.

  Example (for web.xml):

```
<context-param>
   <param-name>csrf.bypass.uri.regex.stk.1</param-name>
```

```
   <param-value>[a-zA-Z0-0]*servlet/param-value>
</context-param>
```

By default, all URIs are in the inclusion list, even if a csrf-include parameter is not provided. You must explicitly specify that a URI is in the exclusion list. If no inclusion list is provided, by default all URIs are considered to be in the inclusion list. Specific URIs can be added to an inclusion list by the application to avoid all URIs being validated for the CSRF token.

By default, the framework provides an exclusion list to bypass CSRF validation for requests for gif, png, css, or js-type files.

5. Most CSRF attacks work just by replicating POST requests into its GET equivalent. Because most applications do not differentiate between POST and GET requests, the attacks usually work. To differentiate between GET and POST requests, in your Struts action definitions, set up the modes in which the validator can operate, using the requestMethodSupported parameter of the action:

   • POST - (default) Only POST requests are allowed.

   If requestMethodSupported is not set or is an unknown value, then it defaults to **POST**.

   • ALL - Both GET and POST requests are allowed.

   Example:

```
<action name="accountTransfer" class="com.AccountTransfer">
  <param name="requestMethodSupported">POST</param>
  <param name="resourceId">AccountTransfer_Action002</param>
</action>
```

Data Handling with the Web UI Framework

Data Type Handling in the Web UI Framework

A consistent method of data type handling is required to validate input boxes on the UI, for defining their entity XML files, and for other tasks. A data type is a data attribute that helps you set constraints on the data, such as acceptable values and what operations may be performed on that data.

The data type is required on the client side of the application for:

- UI field validation (length, size, etc.)

  When you add fields to the screen using the Extensibility Designer, the data types of the new fields help determine the display of the screen.

- UI component display (size, etc.)

Validation can be set up for user events like clicking a button or changing the cursor focus.

With the Web UI Framework, you have the following options for data type handling:

- The default implementation, which lets you continue using the data type handling implementation of the Console JSP, Swing, or RCP UI implementations. Those implementations use the following data type definition files:
  - datatypes.xml (located at <INSTALL_DIR>/repository/datatypes)
  - yfsdatatypemap.xml (located at <INSTALL_DIR>/repository/xapi/template/merged/resource)

  You can customize this default implementation.

- Register the customized implementation of data handling. You can use the web.xml file for this registration.

  The following shows the out-of-the box configuration of the data type-related parameters in the web.xml file. To customize data type handling, you must replace the <param-value> entry with the classpath to the custom Java class, based on its location and package name.

```
<context-param>
   <param-name>scui-datatype-provider</param-name>
   <param-value>
        com.sterlingcommerce.ui.web.platform.dataType.SCUIDataTypeProvider
   </param-value>
</context-param>
```

You can also register the customized implementation by making a Java call to the method SCUIDataTypeHelper.setDataTypeProvider.

The following shows an example of a package for a customized implementation:

```
package com.sterlingcommerce.ui.dataType;
import java.util.Map;
public interface ISCUIDataTypeProvider{
        public Map getDataTypes();
        public SCUIDataType getDataType(StringdataTypeName);
       public SCUIValidationResponse validate(StringdataTypeName, Stringvalue);

        publicbooleanisValid(StringdataTypeName,Stringvalue);
        publicvoidinit();
```

For more information about these packages, refer to the documentation on the interface contracts for data type handling.

The following shows the guidelines for creating a data type using the SCUIDataType class that is used in the above package:

```
package com.sterlingcommerce.ui.dataType;
public class SCUIDataType {

    /** Holds value of property name. */
    private String name;
    /** Holds value of property type. */
    private String type;
    /** Holds value of property size. */
    private Integer size;
    /** Holds value of property decimalDigits. */
    private Integer decimalDigits;
    /** Holds value of property negativeAllowed. */
    private Boolean negativeAllowed;
    .......
    public void setName(String name) {
                   this.name = name;
           }
    public void setType(String type){
        this.type = type;
    }
    public void setSize(int size){
        this.size = new Integer(size);
    }
    public void setDecimalDigits(int decimalDigits) {
                this.decimalDigits = new Integer(decimalDigits);
        }
    public void setNegativeAllowed(boolean negativeAllowed){
        this.negativeAllowed = new Boolean(negativeAllowed);
    }

    .......

    public String getType(){
           return this.type;
```

```
    }

    public boolean isNumeric() {
           return ("NUMBER".equalsIgnoreCase(getType()));
    }
       ........

}
```

To implement the customized Java code, build a jar file that contains the Java class, and then install the jar file using the install3rdparty.sh script.

To implement this customization, rebuild the EAR or WAR file as you did during the installation, and then deploy the application on the application server.

Interface Contracts of the Web UI Framework - Data Type Handling

For more information, refer to the Java API documentation in your installation directory (*<INSTALL_DIR>*/xapidocs/core_javadocs).

Interface Contract	Description	Methods
DataTypeProvider	Implements the ISCUIDataProvider interface, which defines the behavior expected in any implementation of data type handling in an application. DataTypeProvider is plugged in to an application using the context parameter in web.xml: • <param-name> scui-datatype-provider • <param-value> com.application.ApplicationDataTypeProvider	• getDataTypes Returns a map of data types through the merged map of DataType.xml and DataTypeMap.xml. • getDataType(String dataType) Takes the name of the data type and returns the data type object. • validate(String dataTypeName, String value) Validates the value passed against the data type and returns the SCUIvalidationReponse. • isValid(String dataTypeName, String value) Validates the value passed and returns true or false based on the success of the validation. • init Handles initialization.

Assigning Data Types to a Grid Column with the Web UI Framework

You can use the Ext JS JavaScript framework to control the data type of a grid column, instead of using the Properties view of the Designer Workbench. You can program a column data type to depend on the data type of data in corresponding columns of the grid. The data type can be used to determine the column's alignment and sorting behavior.

To define a data type for a grid column, use one of the following config options for the column definition. Work through the order of the list when deciding which config option to use.

1. scuiDataType

   The data type name. If this option is present, the other two config options (bindingData.sFieldConfig.mapping and bindingData.tAttrBinding) are not used.

2. bindingData.sFieldConfig.mapping

   The source binding for the column. An attempt will be made by the application to determine the value of the config. If no data type is found for that value, bindingData.tAttrBinding is used to determine the data type.

3. bindingData.tAttrBinding

   The target binding for the grid column. An attempt will be made by the application to determine the data type for the value of the config.

Once the data type is determined, the following column properties will be defaulted, based on the data type:

- Alignment

  Numbers are right-justified, and dates are middle-justified.

- The type for the store field. The sorting of grid columns is based on the type attribute of the store field config.

  The following list shows the default data types for different data types. For example, if you encounter a number with no decimal digits, it will be stored in the store field as an integer (int).

  - NUMBER (with no decimal digits) - int
  - NUMBER (with decimal digits) - float
  - DATE - date
  - TIME - date
  - DATETIME - date

- renderer

  A renderer is a JavaScript function that can be used to change the text and the look and feel of the application.

- DATE - sc.plat.DateFormatter.getDefaultRenderer('DATE')

  This JavaScript API returns the renderer which would display the date in the format specified for that user.

- TIME - sc.plat.DateFormatter.getDefaultRenderer('TIME')

  This JavaScript API returns the renderer which would display the time in the format specified for that user.

- DATETIME - sc.plat.DateFormatter.getDefaultRenderer('DATETIME')

  This JavaScript API returns the renderer which would display the timestamp in the format specified for that user.

Creating Extra Fields in Grid Stores with the Web UI Framework

The WUF binding framework is used to populate data into a grid. This framework uses the bindingData provided for the grid to create the store. The column configuration is used by the framework to create fields in the store.

Sometimes, more fields (more than the number of columns) may be required in the store for a grid. To add these fields, the bindingData of a grid accepts the fields property.

The following shows an example of a table bindingData object:

```
bindingData: {
  sourceBinding : ["getFlightServiceList:FlightServiceList.FlightService"],
  targetBinding : ["getFlightServiceList:FlightServiceList.FlightService"],
  storeConfig: {
    // ... extra parameters used to create the store
  }
}
```

The following shows an example of a column configuration:

```
{
  dataIndex: '',
  bindingData: {
    sFieldConfig : {
      mapping : "FlightServiceKey"
    },
    tAttrBinding : "FlightServiceKey"
  }
}
```

The configuration of the fields property that is required when adding fields in the store for a grid is shown below:

```
fields: [{
    name: 'fieldName',
    mapping: 'fieldMapping'
}]
```

The following shows an example of the table bindingData object in which the fields property is used:

```
bindingData: {
  sourceBinding : ["getFlightServiceList:FlightServiceList.FlightService"],
  targetBinding : ["getFlightServiceList:FlightServiceList.FlightService"],
  storeConfig: {},
  fields: [{
    name: 'fieldName',
    mapping: 'fieldMapping'
  }]
}
```

Note: If a column with the same name exists in the grid, the definition provided in table binding data under the fields name would be ignored.

Using extensibility, you can add the extn_bindingData property for an existing table (a table which was originally present on the screen, i.e., it was not added to the screen by a user during extensibility).

The following shows an example of an extn_bindingData object that implements the fields property:

```
extn_bindingData: {
  fields: [{
    name: 'fieldName',
    mapping: 'fieldMapping'
  }]
}
```

Customizing Sorting from Multiple Record Fields with the Web UI Framework

You can customize the sorting of table columns that use data that combines multiple fields of a record (for example, by using a renderer). You can do this with both Ext JS 2.2.1 and Ext JS 3.0.

Perform this sort by using the sortType function in the Ext.data.field class. Instead of one item being passed to sortType (value of a field), three items will be passed (value of a field, the record, and the field being sorted). This functionality is provided by overriding the sortData method in the Ext.data.Store class to pass the record and field in addition to the value of the field.

When users are not creating the store for a grid, they can pass sortType as a config option in the bindingData.sFieldConfig property in the columnModel or columns of a grid.

For example, a grid can have the following configuration:

```
columns: [{
  dataIndex: 'airlines',
  bindingData: {
    sFieldConfig : {
      mapping : "airlines",
      sortType: this.sortTypeFn
    }
  }
}]
```

where this.sortTypeFn is defined in the screen JavaScript file as:

```
sortTypeFn: function(val, rec, fld){
              return val + rec.get('airlinenumber');
              // computing the value based on value of field (airlines) and
              // value of 'airlinenumber' field
          }”
```

Supporting Item Quantity Decimal Handling in the Web UI Framework

You can use the Web UI Framework to specify decimal numbers for item quantities. The yfs.install.displaydoublequantity property of yfs.properties indicates whether to support fractional quantities for attributes which belong to the QUANTITY data type. If yfs.install.displaydoublequantity is set to **Y** (the default value), then you can specify decimal numbers for item quantities.

Validating Fields with the Web UI Framework

You can validate fields against certain standards, using the default validation system or your own validation system.

The Web UI Framework provides validation for the following three types of information in the en_US locale:

- E-mail address (using the international accepted standard)
- Telephone number format (locale-specific)
- Credit card number (using the Luhn algorithm)

To validate items, do the following:

1. Register the field attributes that you will be using for validation by implementing the registerFieldAttributes(validationType, attribute) function, using the following arguments:
   - validationType (required)

     Validation type. By default, the application includes validation types for e-mail address, telephone number format, and credit card number.
   - attribute

     XML attributes for validations. An attribute can be registered for multiple validation types.

     Use this function to implement customized validators that you want to plug in to the application.
2. Implement the registerValidators(validationType, validator) function, using the following arguments:
   - validationType (required)

     Validation type.
   - validator (required)

     Validator function for validation type.

The following is an example of how to add validation for a last name:

```
sc.plat.ValidateUtils.registerValidator('LastName', function (value){
    if (value == null || value.length<2) {
        return {status: 2,message: "Last name needs at least two characters" };
    }
    return {status: 1};
});
```

Disabling All UI Fields at One Time with the Web UI Framework

You can use the Web UI Framework to disable all of the screen fields at once, without having to individually disable any field. When you disable a field, you cannot change the data that is in that field. The field becomes read-only. After disabling all of the fields, you can still cut and paste information from those fields, but you cannot submit information from form fields that have been disabled. You cannot disable all of the fields only for look-and-feel purposes.

To disable all of the screen fields at once, use the Ext JS JavaScript method disableFields. A function created from this method has the following properties:

- disable

  Boolean property that determines if all fields on the screen are disabled.

- deep

  Boolean property that determines whether the disable property applies to the immediate children fields of the screen. If this is set to **true**, the disable property applies to all children fields. If this is set to **false**, the disable property applies only to immediate children fields.

- allowCopy

  Boolean property that determines whether a disable method is called for all fields. If this is set to **true**, fields will be marked read-only with an opacity of 0.6. If this is set to **false**, a disable method will be called for all fields.

- disableCSS

  String property that shows the custom css that will be applied if allowCopy is set to **true**.

If the scIgnoreDisable property in a field is set to **true**, that field will ignore the disableFields method.

Checking for Screen Changes in the Web UI Framework

In screens created using the Web UI Framework, the application can take actions that are based on whether a screen field changed. For example, if you open a screen to modify a field, and you end up not modifying that field, you could program the application not to submit information from that screen to a server when you close the screen.

The isDirty method checks all of the fields of a screen to see if they have changed. Each editable field also has an isDirty method, so you can program the application to take actions based on whether a particular field changed.

Screen changes are also monitored using a dirtystatechange event. Whenever a field is modified on a screen, the dirtystatechange event is fired on the screen. In the following example, the **Save** button is enabled whenever a field on a screen is modified:

```
Screen.addListener('dirtystatechange',function(scr, isDirty)
   {
   savBtn.disable(!isDirty);
   },
this);
```

Configuring a Data Source with the Web UI Framework

To work with a data source, you must first configure it using the Configure Data Sources dialog box.

The Web UI Framework does not use XML binding. The Configure Data Sources dialog box works with only JSON data sources.

1. Make sure the Data tab is showing, and not the Palette or Files tab.

2. Click the button that is just to the right of the dropdown arrow.

   The Configure Data Sources dialog box appears.

3. Configure the following items:

   • Type of data source (input or output)
   • Data source

     The data source directory is the directory containing JSON data source files, which can be provided to application developers. With the Web UI Framework, a tool is provided for generating JSON data sources from XAPI XML and XSD definitions.

   • Path to data source directory
   • Namespace, elements, and attributes

4. Click the **Finish** button.

   The data source is configured. This initializes the bindingData property of the widget that is using the data source. You can also specify binding by creating or editing this property in the Properties view.

Adding a Data Source with the Web UI Framework

To work with a data source, you must first configure it using the Configure Data Sources dialog box.

1. Make sure the Data tab is showing, and not the Palette or Files tab.
2. Make sure that you are on the screen where you want to add a data source.
3. Click on the down arrow to select a data source.
4. Add the data source to the screen.