Sterling Web™

Customization Guide

Release 9.0

Last updated in HF1

May 2010

Sterling Commerce
An IBM Company

Contents

Customizing Sterling Web: An Overview

The Sterling Web™ application can be customized based on the business requirements of an enterprise. Organizations may want to make changes in the way information is displayed in the Sterling Web user interface (UI), or add a new field. This topic provides an overview of the types of customizations possible in the Sterling Web application.

The Sterling Web application uses the Apache Struts 2 framework for page construction and request management between pages. Sterling Web customizations can be performed by selectively overriding the action definitions. Struts 2 action definitions bind together a collection of resources required to fulfill any type of request from the Web. Custom action definitions can selectively overlay a portion of the action namespace for users affiliated with a specific enterprise or accessing a specific storefront. Within this context, the users affiliated with other enterprises or interacting with different storefronts will, however, not be able to view the overlaid action definitions. The overlaid action definitions can bind the existing Sterling Web resources with new resources to add new functionalities to the application.

The following components can be customized in Sterling Web:

- Struts extensions — Defining new Struts and overriding the existing Struts.
- Mashup extensions — Customizing the input XML and the output template of an API call. Additionally, a user can perform the following functions with mashups:
  - Define new mashups
  - Override the existing mashup using override extensibility
  - Extend the mashups using differential extensibility

  For more information about how to override extensibility and extending mashups using differential extensibility, refer to the *Selling and Fulfillment Foundation: Customizing the Web UI Framework Guide*.

  Note: A user can extend the mashups specific to a storefront by defining the mashup in the customized struts action and including the corresponding mashup file to the customization folder created by the user. However, a user can extend mashups across multiple storefront by following the approach specified in the *Selling and Fulfillment Foundation: Customizing the Web UI Framework Guide*. For more information about extending mashups, refer to the *Selling and Fulfillment Foundation: Customizing the Web UI Framework Guide*.
- JSP — Creating new JSPs and overriding the existing JSPs.
- Themes — CSS and image files pertaining to the application's look and feel are organized in a directory hierarchy, with the theme name as the root. By default, each storefront is assigned a theme. The CSS files pertaining to a theme are used in construction of pages for users accessing a storefront. For more information about associating a theme with a storefront and controlling the look and feel by using CSS, refer to the *Sterling Web Implementation Guide*. For information about theme, CSS files and understanding the Sterling Web user interface, refer to the *Sterling Web User Interface Architecture Guide*.

For information about customization basics, refer to the *Selling and Fulfillment Foundation: Customization Basics Guide*.

If a user adds a new field in the JSP, the user may have to extend the database table to add a corresponding database column. In such a scenario, the database is extended to include a corresponding column. The new column that is added in the database must be exposed to the corresponding APIs. For more information about extending the database, refer to the *Selling and Fulfillment Foundation: Extending the Database Guide*.

Action Definition Customization Checklist

When customizing the action definitions that are a part of Sterling Web, follow the sequence in which the tasks are listed in the following checklist.

1. Create the root folder for the customization project. For more information about how a user can structure the folders for customizations, see the "Customization Examples" topic.
2. Under the Customization project create a customization folder, say Cust1.
3. Determine the Struts file you want to customize. To identify the files that require customization, it is important to know the relevant resources such as the action name, mashup XML, resource id, binding file and JSP file that are defined as part of the corresponding Struts file.
4. As part of customization, you can either introduce a new Struts file in the customization project or modify the existing Struts file by copying it to the customization project with a different name.
5. To introduce a new Struts file perform the following steps:
    a. Add a new action name.
    b. Add a new JSP page.
    c. Add a new mashup XML file.
    d. Add a new XML binding file. The binding files bind the XAPI variables with the JSP components and JAVA attributes with the API attributes.

    To modify the existing Struts file perform the following steps:

    a. Copy the existing action name to the customization project and modify it.
    b. Copy the existing JSP page to the customization project and modify it.
    c. Copy the existing mashup XML file to the customization project and modify it.
    d. Copy the existing XML binding file to the customization project and modify it.

    Based on the requirement, you must either add or update the appropriate files defined in the customized Struts file.
6. Include the newly added or modified Struts file into `custom_struts.xml` file placed under the customization project.
7. Include the `custom_struts.xml` file in to the `swc_struts.xml.sample` file located in the `<INSTALL_DIR>/repository/eardata/swc/extn` folder.
8. Rename the `swc_struts.xml.sample` file to `swc_struts.xml` file.
9. Build the JAR file of the customization project.
10. Install the JAR file using the installService.
11. Create the EAR and deploy the EAR.

Theme and Logo Customization Checklist

1. Create the root folder for the customization project. For more information about how a user can structure the folders for customizations, see the “Customization Examples” topic.
2. Under the customization project, create a customization folder, say Cust1.
3. Create a new theme, say, theme1 or assign an existing theme to the enterprise that must be customized.
4. Add the customized CSS, js, image and logo files to the customization folder. Ensure that the files are placed in the appropriate folder. For example, the CSS file must be placed in the `customization/src/main/webapp/swc/css/theme` folder. For more information about how a user must structure the folders for customizations, see the “Customization Examples”.
5. Build the JAR file of the customization project.
6. Install the JAR file using the installService.
7. Create the EAR and deploy the EAR.

Storefront Specific Customizations

You can perform customization for a storefront. To perform storefront specific customizations, the storefront name must be prefixed to the namespace in the Struts action definition, as follows:

```
<package name="S1.profile.user" namespace="/S1/profile/user" extends="user">
```

Here, S1 is name of the storefront.

Customizations Across Multiple Storefronts

You can perform customizations across storefronts and themes by changing the out-of-the-box settings. If multiple storefronts have the same default theme, you can perform only one set of customization for that theme. The customizations will then be applied to all the storefronts that have the same theme as their default theme.

To perform customizations across multiple storefronts, the storefront's theme name must be prefixed to the namespace in the Struts action definition is as follows:

```
<package name="T1.profile.user" namespace="/T1/profile/user" extends="user">
```

Here, T1 is the name of the theme and the Struts action definition of the namespace is applicable to all the storefronts associated with the theme, T1.

In this case, the storefronts associated with a common theme will have the same look and feel. If the user wants a different look and feel for each storefront, customization has to be done at the storefront level.

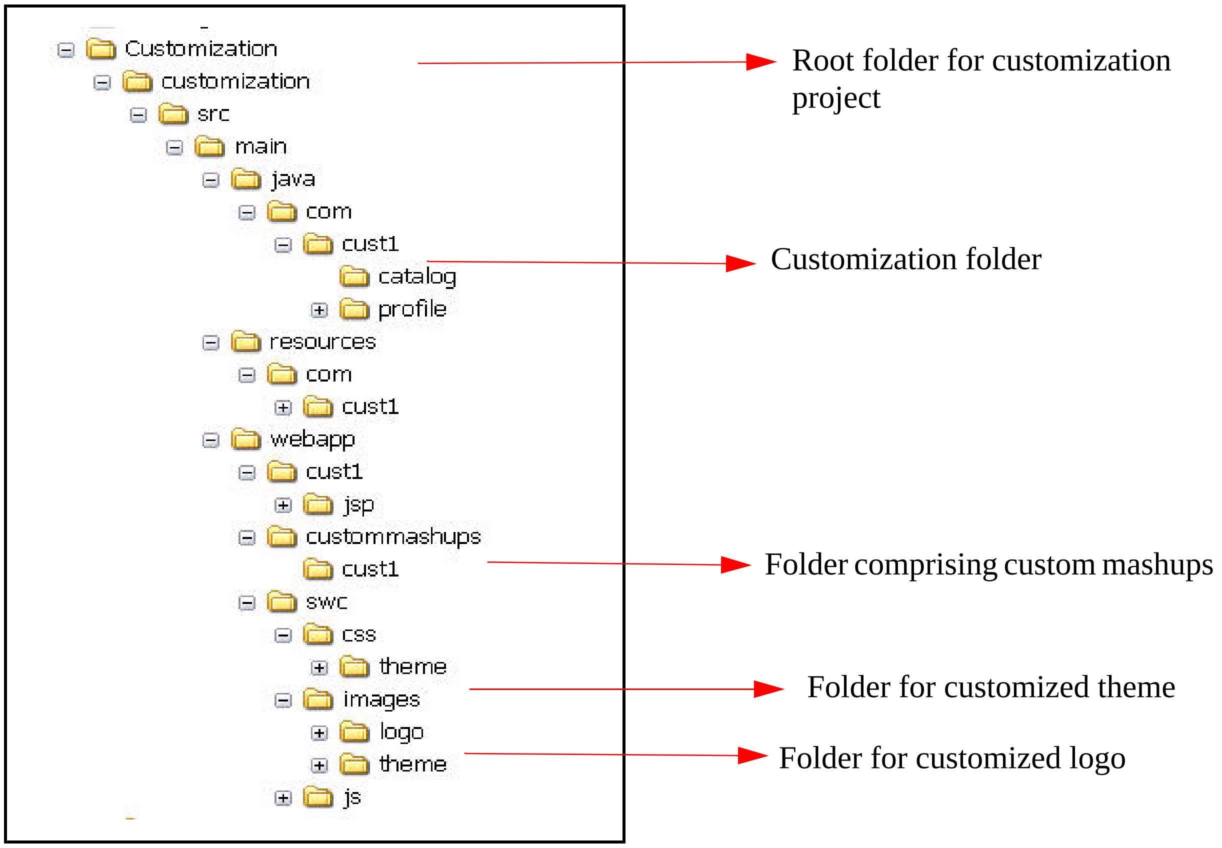
Note: Do not modify the action definitions, JSPs, mashup definitions, and files that are provided out-of-the-box by Sterling Web directly. Ensure that you copy them to the customization folder and then modify them.

The Struts 2 framework processes the request based on the fall back mechanism by first prefixing the storefront name to the namespace. If this action exists, the Struts framework dispatches the process to the Strut action definition. If this action does not exist, the Struts 2 framework prefixes the storefront theme name to namespace. If this action exists, the Struts framework dispatches the process to the Struts action definition. If this action also does not exist, the Struts 2 framework prefixes the default name, swc to the namespace and dispatches it to the Struts action definition.

Customization Examples

This topic provides few example of customization to provide a better understanding of the customization process. Apart from the examples provided here, a user can perform customizations based on the individual's requirements.

A user may structure the folders for customizations as shown in the following figure:



The following table lists the files and the corresponding location required for customization:

Files	Location
Mashup	customization/src/main/webapp/custommashups/cust1
Struts	customization/src/main/resources/com/cust1
JSP	customization/src/main/webapp/cust1
Class	customization/src/main/java/com/cust1

Note: All the Sterling Web Struts action definitions are included in the `swc_struts.xml` file.

All the customization examples in this topic are performed for a storefront named, ABC.

Theme and Logo Customization for a Storefront

By default, Sterling Web provides themes. A user can, however, create a new theme, or assign an existing theme to an enterprise through the Applications Manager. For more information about defining a theme for an enterprise, refer to the *Selling and Fulfillment Foundation: Application Platform Configuration Guide*. For information about theme CSS, exploring a theme, and changing theme-related styles, refer to the *Sterling Web User Interface Architecture Guide*.

When a default theme is specified for a storefront, the corresponding theme `.css` file should be stored in the `\swc\css\theme\<default theme name of storefront>\theme-1.css` folder that may include other `.css` files. If a theme is not specified for a particular storefront, the default theme provided as factory setup is used as the theme for that storefront. This default theme corresponds to the `theme-1.css` file that is located in the `\swc\css\theme\` folder.

Note: Theme files corresponding to the themes provided by Sterling Selling and Fulfillment are not provided by Sterling Web.

A user can follow the approach mentioned below to associate a theme and logo with a storefront:

- Specify the theme for a storefront by saving the corresponding theme `.css` file in the `<INSTALL_DIR>repository\eardata\swc\war\swc\css\theme\<default theme name of storefront>\` folder.
- Specify the logo that is associated with a storefront theme by saving the `logo-<default theme name of storefront>.gif` file in the `<INSTALL_DIR>repository\eardata\swc\war\swc\images\logo\<Organization Code of the storefront>\` folder. The image associated with the logo must be 253 X 37 pixels.
- Specify the default logo that will be used in the absence of a theme-based logo for a storefront, by saving the `logo.gif` file in the `<INSTALL_DIR>repository\eardata\swc\war\swc\images\logo\<Organization Code of the storefront>\` folder. The image associated with the logo must be 253 X 37 pixels.

Ensure that you create and deploy the SWC EAR. For more information about deploying EAR and WAR files, refer the *Selling and Fulfillment Foundation: Installation Guide*.

Note: The theme name assigned to a storefront and the storefront ID are case sensitive.

Alternatively, a user can customize the theme and logo for a storefront by adding the appropriate files in the customization project by performing the steps mentioned below:

1. Create the root folder for customization say, ABC_customization.
2. The name of the customized theme for a storefront ABC can be abc_cust.
3. In the following folders, replace the `{custom_theme}` attribute with the storefront theme name:
   - `customization/src/main/webapp/swc/css/theme/{custom_theme}`
   - `customization/src/main/webapp/swc/js/theme/{custom_theme}`
   - `customization/src/main/webapp/swc/images/theme/{custom_theme}`

- `customization/src/main/webapp/swc/images/logo/{storefront_id}/logo-{custom_theme}.gif`

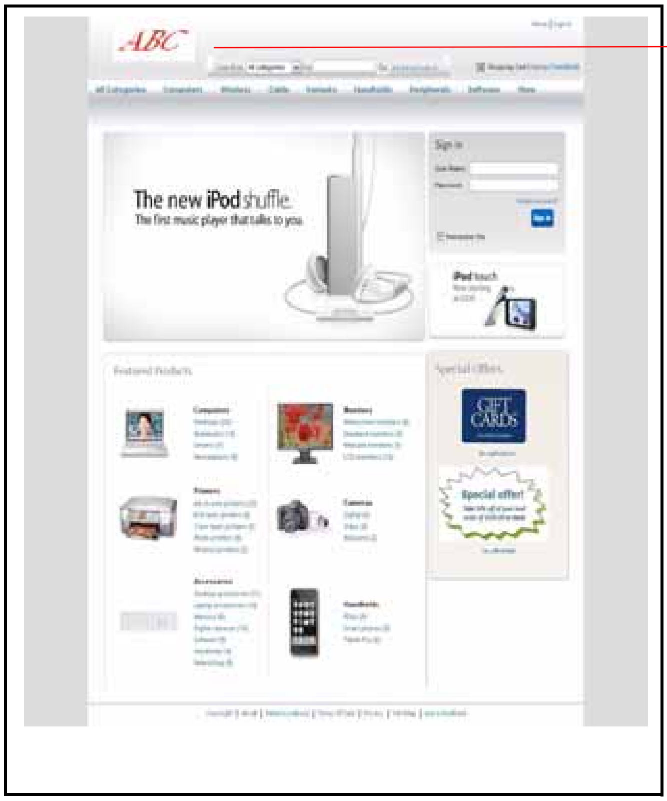
The following table lists the files and the corresponding location required for implementing the abc_cust theme and logo customization:

File Name	Location
theme-1.css	customization/src/main/webapp/swc/css/theme/abc_cust
theme.js	customization/src/main/webapp/swc/js/theme/ abc_cust
logo-abc_cust.gif	runtime/src/main/swc/images/logo/ABC

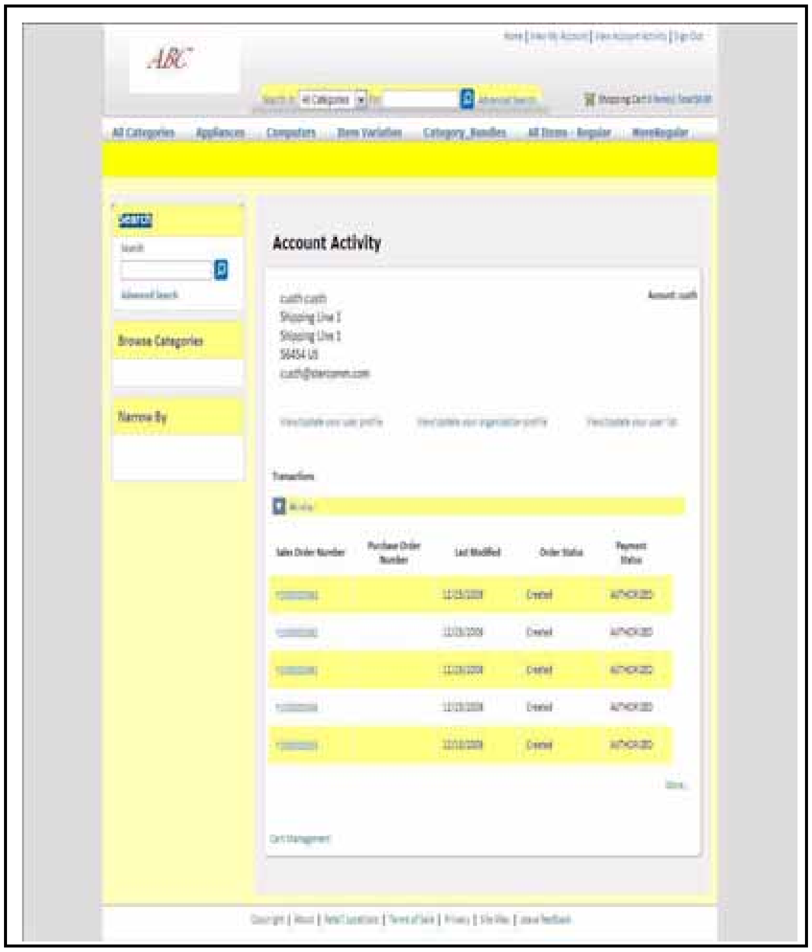
All the images pertaining to the theme are stored in the `customization/src/main/webapp/swc/images/theme/abc_cust` folder.

4. Ensure that you deploy the customized JAR by performing the steps mentioned in the "Postcustomization Deployment" topic.

The following figure displays the customized logo for ABC storefront:



The following figure displays the customized theme for ABC storefront:


Account Activity

custh custh
Shipping Line 1
Shipping Line 1
56454 US
custh@stercomm.com

Account: custh

Transactions

Sales Order Number	Purchase Order Number	Last Modified	Order Status	Payment Status
[illegible]		[illegible]	Created	AUTHORIZED
[illegible]		[illegible]	Created	AUTHORIZED
[illegible]		[illegible]	Created	AUTHORIZED
[illegible]		[illegible]	Created	AUTHORIZED
[illegible]		[illegible]	Created	AUTHORIZED

Adding a New Flow and Functionality

A user can add a new flow or functionality to the Sterling Web application. For example, a user can add a new hyperlink on the Account Activity page to view a list of all the currencies supported by Sterling Web.

For this, the PortalHomeAddress JSP page must be customized to display the View/Update Customer Currency hyperlink. A user must override the action definition of the PortalHomeAddress JSP page to redirect the control to the customized page. Whenever a user clicks the View/Update Customer Currency hyperlink, the CustomerCurrenciesAction.java action is invoked which in turn, calls the getCurrencyList API. This API fetches the results and the CustomerCurrencies.jsp displays the customized page with the list of currencies supported by Sterling Web.

The steps to customize the Account Activity page are as follows:

1. Create the root folder for customization say, ABC_customization.
2. The URL of the Account Activity page is as follows:

   `http://<hostname>:<port>/swc/home/portalHome.action?sfId=<StorefrontID>`

Here, the portalHome.action renders the Account Activity page and /swc/home/ is the namespace. The `swc_struts.xml` file contains the Struts file names used by Sterling Web. Based on the namespace and the action name the corresponding Struts file can be determined by referring the `swc_struts.xml` file. The `home-struts.xml` file comprising the portalHome.action is located in the `/main/resources/com/sterlingcommerce/webchannel/home` folder.

3. The `home-struts.xml` file must be overridden to redirect the control to the customized Account Activity page.

   The customized `cust1_struts` file is as follows:

```
<package name="cust11" extends="home" namespace="/ABC/home">
            <action name="portalHomeUserAddress"
class="com.sterlingcommerce.webchannel.home.PortalHomeUserAddressAction">
            <param name="mashups">getAddressBook</param>
            <param
name="xmlBindingFile">PortalHomeUserAddress_binding.xml</param>
            <param name="resourceId">/swc/portalHome</param>
            <result
name="success">/cust1/jsp/home/portalHomeUserAddress.jsp</result>
        </action>
</package>
```

4. Include the Struts action definition for the customized CustomerCurrenciesAction action in the `cust1_struts` XML file as follows:

```
<package name="cust1" namespace="/ABC/profile/user" extends="wcDefault">
<action name="getCustomerCurrencies"
            class="com.cust1.profile.customer.CustomerCurrenciesAction">
            <param name="mashups">customer_currencies</param>
            <param name="resourceId">/swc/profile/ManageUserList</param>
           <!--<param name="xmlBindingFile">UserList_binding.xml</param>-->
            <result
name="success">/cust1/jsp/profile/user/CustomerCurrencies.jsp</result>
        </action>
</package>
```

   Here, cust1 is the name of the customization. The `cust1_struts.xml` file comprises the namespace and action names that have been added as part of the new functionality.

5. Include the files defined in the Struts action definition. The following table lists the files and their locations for overriding the existing functionality:

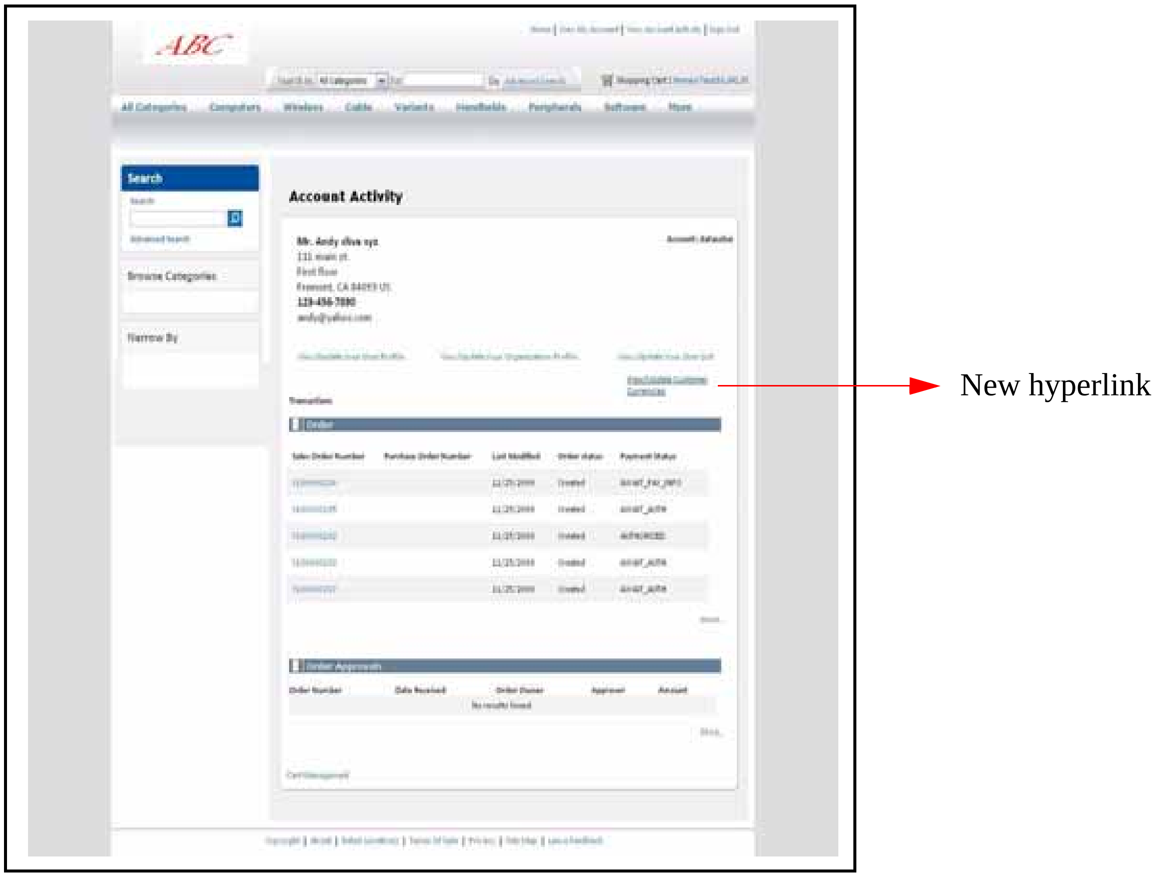
File Name	Location
CustomerCurrencies.jsp	/customization/src/main/webapp/cust1/jsp/profile
portalHomeUserAddress.jsp	/customization/src/main/webapp/cust1/jsp/home
cust1_mashup.xml	/customization/src/main/webapp/custommashups/cust1/profile
CustomerCurrenciesAction.java	/customization/src/main/java/com/cust1/profile/customer
cust1_struts.xml	/customization/src/main/resources/ com/cust1/profile/customer
DCL.xml	/customization/src/main/resources
custom_struts.xml	/customization/src/main/resources

6. The customized struts file, `cust1_struts.xml` pertaining to the new flow must be included in the `custom_struts.xml` as follows:

```
<struts>
<include file="com/cust1/profile/customer/cust1_struts.xml"/>
</struts>
```

7. Deploy the customized JAR by performing the steps mentioned in the "Postcustomization Deployment" topic.

The following figure displays the new hyperlink that can be used to view a list of all the currencies supported by Sterling Web added as part of customization on the View Account Activity page of ABC storefront:



Overriding an Existing Functionality

Sterling Web enables a user to override an existing functionality. A user must identify the files for customization, copy them to the respective customization folders, and modify them appropriately. A user can add a new field in the user interface, extend the mashup to include a new attribute, modify the validations in the user interface, and so on. For example, a user can add the Effective Start Date and Effective End Date of the product on the Item Details page for a storefront, ABC. To perform this task, the user must override the action definition of the Item Details page to redirect the control to the customized page. The new mashup definition must have the Effective Start Date and Effective End Date fields. These fields are passed in the output of the getCompleteItemList API and displayed in the customized JSP.

The steps to customize the Item Details page are as follows:

1. Create the root folder for customization say, ABC_customization.

2. The URL of the Item Details page is as follows:

   `http://<hostname>:<port>/swc/catalog/itemDetails.action?sfId=<StorefrontID>`

   Here, `itemDetails.action` renders the Item Details page and `/swc/catalog/`is the namespace.

   The swc_struts.xml file contains the Struts file names used by Sterling Web. Based on the namespace and the action name, the corresponding Struts file can be determined by referring the swc_struts.xml file. The `catalog-struts.xml` file comprising the itemDetails.action is located in the `/main/resources/com/sterlingcommerce/webchannel/catalog` folder.

3. The `itemDetails.action` is defined in the action definition specified in the `catalog-struts.xml` file The action definition must be overridden to redirect the control to the customized JSP page.

4. The new action definition in the `cust1_struts.xml` file is as follows:

```
<struts>
<package name="abc.catalog" namespace='/ABC/catalog' extends="catalog">
  <!-- action for item Details -->
  <action name="itemDetails"
class="com.cust1.catalog.Custom_ItemDetailsAction">
         <param name="mashups">custom_itemDetails</param>
         <param name="resourceId">/swc/catalog/catalogBrowsing</param>
         <param
name="xmlBindingFile">com/cust1/catalog/Custom_ItemDetails_binding.xml</par
am>
  <param name="numOfItemsPerPromotion">2</param>
         <param
name="validationClass">com.cust1.catalog.Custom_ItemDetailsValidation</para
m>
         <result
name="success">/cust1/jsp/catalog/Custom_itemDetails.jsp</result>
  </action>
  <!-- End -->
  </package>
</struts>
```

   Here, cust1 is the name of the customization.

5. Include the files defined in the struts action definition, in the appropriate folder structure.

   The following table lists the files and their locations required for overriding the existing functionality:

File Name	Location
Custom_itemDetails.jsp	/customization/src/main/webapp/cust1/jsp/catalog
cust_catalog.xml	/customization/src/main/webapp/custommashups/cust1
Custom_ItemDetailsAction.java	/customization/src/main/java/com/cust1/catalog

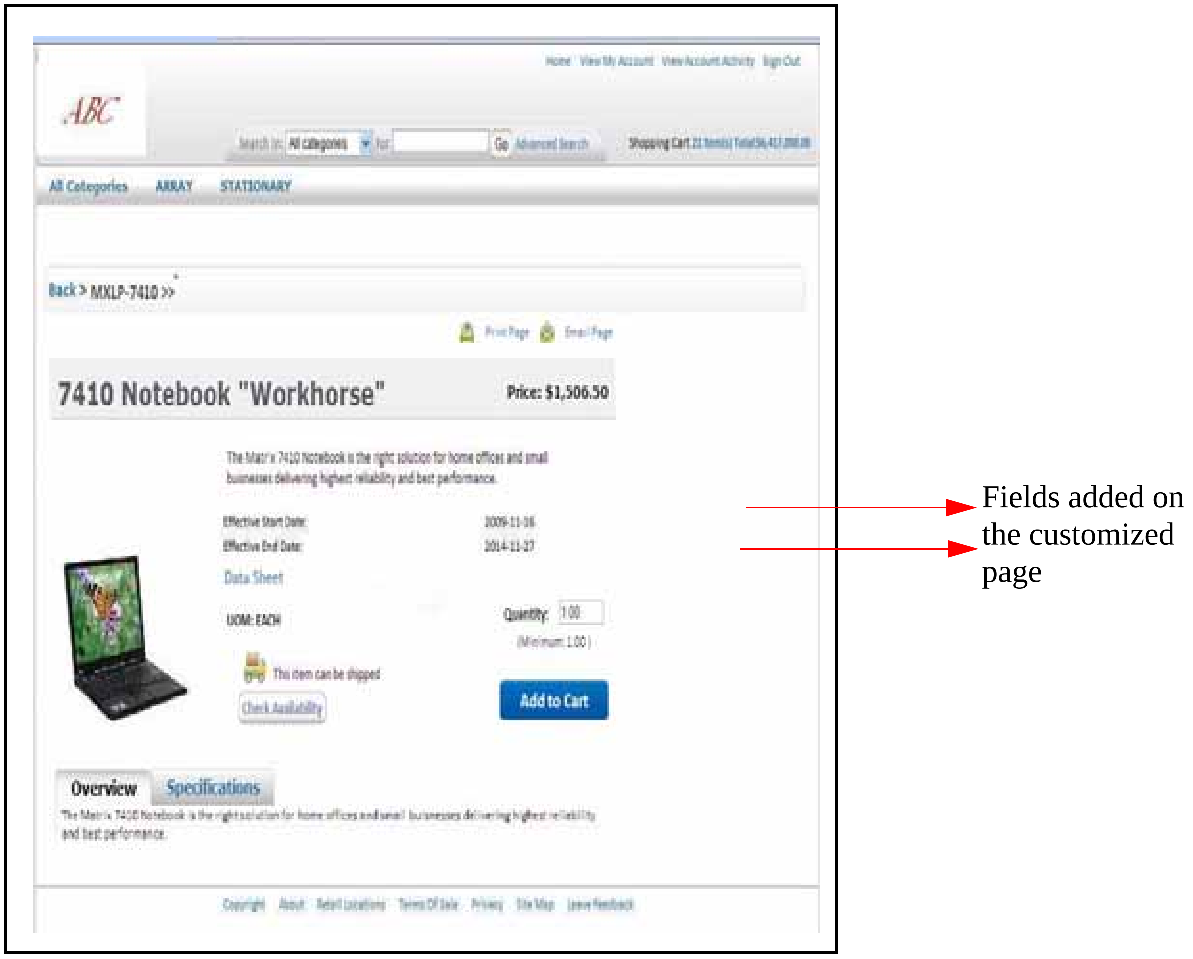
File Name	Location
Custom_ItemDetailsValidation.java	/customization/src/main/java/com/cust1/catalog
cust1_struts.xml	/customization/src/main/resources
Custom_ItemDetails_binding.xml	/customization/src/main/resources/com/cust1
DCL.xml	/customization/src/main/resources
custom_struts.xml	/customization/src/main/resources

6. The customized struts file, `cust1_struts.xml` pertaining to the new flow must be included in the `custom_struts.xml` as follows:

   ```
   <struts>
   <include file="com/cust1/catalog/cust1_struts.xml"/>
   </struts>
   ```

7. Deploy the customized JAR by performing the steps mentioned in the "Postcustomization Deployment" topic.

   The following figure displays the customized Item Details page added as part of the new functionality for ABC storefront:



Back > MXLP-7410 >>

7410 Notebook "Workhorse" Price: $1,506.50

The Matrix 7410 Notebook is the right solution for home offices and small businesses delivering highest reliability and best performance.

Effective Start Date:	2009-11-16
Effective End Date:	2014-11-27

Data Sheet

UOM: EACH

Quantity: 1.00 (Minimum: 1.00)

This item can be shipped

The Matrix 7410 Notebook is the right solution for home offices and small businesses delivering highest reliability and best performance.

Customizing the Alternative View of the Product List page

By default, Sterling Web enables a user to compare the products on the Product List page by dragging and dropping them to the Drag to Compare basket. However, an end-user can customize the Product List Page.

By performing the following steps a user will be able to select the products on the Product List page and compare them by clicking the **Compare** button:

1. Determine the name of the storefront for which you want to associate the alternative view.
2. Copy the `alternate_catalog_flow_struts.xml.sample` file located in the `<INSTALL_DIR>/repository/eardata/swc/war/WEB-INF/classes` folder and paste it in the `customization/src/main/resources` folder created by the user.

3. Rename `alternate_catalog_flow_struts.xml.sample` to `alternate_catalog_flow_struts.xml.`

   Note: If this file is not present in the folder, run the `buildear` script to generate this file.

4. Edit `custom_struts.xml` to include the `alternate_catalog_flow_struts.xml` file as follows:

   ```
   <include file="alternate_catalog_flow_struts.xml"/>
   ```

5. In the `alternate_catalog_flow_struts.xml`, replace the `_replace_with_storefront_name_` attribute with the name of the storefront for which you want to configure the alternative view.

6. Perform the configurations required to set up the storefront. For more information about setting up a storefront, refer to the *Selling and Fulfillment Foundation: Application Platform Configuration Guide.*

7. Deploy the customized JAR by performing the steps mentioned in the "Postcustomization Deployment" topic.

   Note: If multiple storefront have to be associated with the alternative page, the storefront theme name must be used in the Struts definition.

Other Customizations

In addition to the customizations described in the previous sections, users can perform other customizations of their choice. To perform any type of customization, a user must replace the `{custname}` attribute in the following folders with the appropriate folders for customization:

- `/customization/src/main/webapp/{custname}/css/..`
- `/customization/src/main/webapp/{custname}/jsp/..`
- `/customization/src/main/webapp/custommashups/{custname}/..`
- `/customization/src/main/webapp/{custname}/images/..`
- `/customization/src/main/webapp/{custname}/js/..`

The following table lists the file types and their corresponding locations, as provided by Sterling Web by default:

File	Location
Java class files	/customization/src/main/java/com/{vendor-name}/{customization name}..
Binding files	/customization/src/main/resources/com/{vendor-name}/{customization name}..
Resource bundle	/customization/src/main/resources /com/{vendor-name}/{customization name}..
Struts configuration file	/customization/src/main/resources /com/{vendor-name}/{customization name}..
Struts validation file	/customization/src/main/resources /com/{vendor-name}/{customization name}..
Custom Struts file	/customization/src/main/resources

The Sterling Web Java classes and class loader resources required to perform customization reside in the `com.sterlingcommerce.webchannel` package. Ensure that the nomenclature of the folder structure avoids naming conflicts. It is recommended that the customization classes have a common root package similar to `com.{vendor-name}.{customization name}` as specified in Sterling Web.

Postcustomization Deployment

After the customizations are performed, the user must deploy the changes in the application by performing the following steps:

1. Generate the customization JAR file.
2. Install the customization JAR file using the installService utility. In the customization JAR file, ensure that the customized files have been placed in the appropriate locations, as described in the tables below.
3. Deploy the customizations by creating and deploying the SWC EAR file.

For example, if you are adding or overriding a functionality, the customized files must be placed in the appropriate locations as described in the following table:

Files	**Location**
3rd party jar files	The path generated by install3rdparty utility. For more information about the install3rdparty utility, refer to the *Selling and Fulfillment Foundation: Installation Guide*.
Java Server Page	\swc\files\repository\eardata\swc\war\\`<user defined folder>`
Javascript	\swc\files\repository\eardata\swc\war\\`<user defined folder>`
Mashup definitions	\swc\\\files\repository\eardata\swc\war\mashupxmls\webchannel\\`<user defined folder>`
Static image files	\swc\files\repository\eardata\swc\war\\`<user defined folder>`

If you are customizing the theme and logo, the customized files must be placed in the appropriate locations as described in the following table:

Files	**Location**
Image for logo	\swc\files\repository\eardata\swc\war\swc\images\logo
CSS files	\swc\files\repository\eardata\swc\war\swc\css\theme
JS files	\swc\files\repository\eardata\swc\war\swc\js\theme

The `DCL.xml` file is the common file required for all customizations. This file is included in the `\swc\jars\` path in the customization JAR. The DCL.xml file must have an entry of the JAR file comprising the customized Java classes and resources. The name of the JAR files is case-sensitive. For example, the customization JAR file generated for the storefront ABC will be `ABC_CUST.jar` and the jar file comprising the customized Java classes and resources will be `abc_cust.jar`. The user must generate `abc_cust.jar` file and add it to the `\swc\jars\abc_cust\<user_defined_version>\` path in the `ABC_CUST.jar`.

In the above example, the `DCL.xml` file has the entry for the `abc_cust.jar` as follows:

```
<dcl>
<vendor>
<name>abc_cust/<user_defined_version>/abc_cust.jar</name>
<target>DCL|APP</target>
</vendor>
</dcl>
```

To deploy the customizations to the application, the user must perform the following steps:

1. Run the following command to generate the customization jar file. Ensure that the customization jar comprises the `DCL.xml` file:

   ```
   ant -f <customization script file> <target>
   ```

2. Run the following command to install the customized jar file:

   For Windows:

   `InstallService.cmd <location of customized jar>`.

   For Linux or Unix:

   `InstallService.sh <location of customized jar>`.

3. Navigate to the <`INSTALL_DIR`>`/repository/eardata/{custname}/extn` folder and rename `struts.xml.sample` as `struts.xml`.

4. Add the entry of `custom_struts.xml` in the `struts.xml`file. This must be done for all the new customized action definitions.

   ```
   <struts>
   <include file="sic_struts.xml"/>
   <include file="scuiimpl_struts.xml"/>
   …
   <include file="custom_struts.xml"/>
   …
   </struts>
   ```

5. Build and deploy the SWC EAR file. For more information about deploying EAR and WAR files, refer the *Selling and Fulfillment Foundation: Installation Guide*.

Customizing Validations

The Sterling Web™ application enables a user to customize user interface-level validations and data validations. Organizations may want to either create new validations or modify existing validations. This topic provides an overview of customizing the validations in the Sterling Web application. For example, a user can mandate a field, create a combination of fields, and so on. For information about data validations and user interface validations, refer to the *Sterling Web Implementation Guide*.

Customizing UI Validations

UI validations can be customized based on a user's requirement, as follows:

- Adding Struts Validators
- Customizing Action-Specific Validation and Page-Specific Validation
- Customizing E-Mail Validation
- Customizing Phone Number Validation

Adding Struts Validators

By default, the Struts client-side validation framework provides certain validators. For more information about using the Struts validation framework for user interface validations, refer to the *Sterling Web Implementation Guide*. New validators can be added in Sterling Web to perform specific validations by providing the appropriate entry for the new validators in the `<swc_war_file>/WEB-INF/classes/validators.xml` file and the corresponding implementation class. The new validators can be used for validating any field in any page in the Sterling Web application.

Customizing Action-Specific Validation and Page-Specific Validation

By default, Sterling Web provides UI validations. A user can customize UI validations for a specific page.

A user can perform the following tasks as part of customizing UI validations for a page:

- Customizing action-specific validation for an existing page: Action-specific validations can be customized by creating custom action- specific validation class. The classname must be named appropriately and added in the Struts definition for the corresponding action. For more information about action-specific validation definition of class, refer to the *Sterling Web Implementation Guide*.
- Adding new fields in a form: JSP customization must be done for add new form fields in a page. In such a scenario, a user may have to customize the validation. This can be done by adding action-specific validation class, as explained previously.
- Adding new fields by extending the database table: A user can extend an existing database table to add new columns. If a new column is used as a form field in a Sterling Web JSP, the user must customize the validation for that field. However, to be able to use the Ajax validation framework for the new fields, appropriate data type must created for the new database column. For more information about extending data type files, refer to the *Selling and Fulfillment Foundation: Extending the Database Guide*.

- Customizing UI validation error messages: The error messages displayed in the UI are displayed after localization. These messages can also be customized. For more information, about customizing the Sterling Web application, refer to the *Sterling Web Localization Guide*.

Customizing E-Mail Validation

By default, Sterling Web provides the swc.email.validator.regex system property. The value of this property is a regular expression that is used to validate the e-mail addresses of the users. To customize e-mail validation, the `swc.email.validator.regex system` property must be overridden with the regular expression against which the e-mail validation must be performed.

Customizing Phone Number Validation

By default, Sterling Web validates all the phone number fields based on US format, that is, +xxx (yyy) zzz-zzzz, where xxx is the Country code, yyy is the Area or City code, and zzz-zzzz is the Local number. The validation logic for this can be customized based on a user's requirement. Sterling Web provides the `swc.validator.phone` system property that defines the implementation class to be used for phone number validation. The default implementation class for this property is com.sterlingcommerce.webchannel.core.validators.WCDefaultPhoneNumberFormatter. For customization, users can create a custom implementation class and override the `swc.validator.phone` system property to point to the new class. To perform phone number validation, a class must implement the com.sterlingcommerce.webchannel.core.validators.IWCPhoneNumberFormatter interface.

Customizing Data Validation

A user can customize data validation rules by creating new validation rules and deleting existing validation rules. Ensure that data validation is enabled for Sterling Web in the `web.xml` file located in the `EARFILE/WARFILE/WEB-INF` folder.

To customize data validation rules:

1. Include the following property files in the class path of the application server. These files are an extension to the property files provided by Sterling Web:
   - swc_InputType_ValidationRules_extn.properties
   - swc_ParamValue_ValidationRules_extn.properties

   For more information about the property files with default validation rules provided by Sterling Web, refer to the *Sterling Web Implementation Guide*.
2. Define customized validation rules by providing entries in the property files mentioned previously. or information about implementing data validations, refer to the *Sterling Web Implementation Guide*.
3. Restart the application server.

Note: By default, Sterling Web provides validation rules specific to the en_US locale. If a user wants to support a different locale, the user must add characters pertaining to that locale. Validation rules are not specific to a locale and are generic and common across all the locales.

Index