Sterling Store Associate Mobile

Customization Guide

3.2.00

Contents

Copyright

Extensibility in the Web UI Framework

Extending Mashups Using Override Extensibility in the Web UI Framework

Using IBM® Sterling Store Associate Mobile 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>.

   For example, you might have the following out-of-the-box mashup:

```
<mashup id='demoapp-stk-getFlight' transactional='true' description="Flight
Fetch mashup"
                        mashuptype="XAPI">
     <classInformation
name="com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIMashup" />
     <API Name="getFlight">
           <Input>
                  <Flight FlightKey="" />
           </Input>
           <Template>
                  <Flight FlightName="" />
           </Template>
     </API>
</mashup>
```

   To use differential extensibility, you could create an incremental/differential mashup as shown below in the following example override_mashup.xml file in a /myapps/override directory with the same mashup ID:

```
<mashup id='demoapp-stk-getFlight' mashuptype="XAPI">
     <classInformation
name="com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIMashup" />
     <API Name="getFlight">
           <Template>
                     <Flight  OrganizationCode="" />
           </Template>
     </API>
</mashup>
```

   If you use the incremental load method `loadIncrementalMashupXml(/myapps/override/override_mashup.xml, ...)`, then the inner elements in the override_mashup.xml file are merged into the original mashup, and it will behave as though the mashup was coded as follows:

```
<mashup id='demoapp-stk-getFlight' transactional='true' description="Flight
Fetch mashup"
                        mashuptype="XAPI">
     <classInformation
name="com.sterlingcommerce.ui.web.platform.mashup.SCUIXAPIMashup" />
     <API Name="getFlight">
           <Input>
                  <Flight FlightKey="" />
           </Input>
           <Template>
                  <Flight FlightName="" OrganizationCode="" />
           </Template>
     </API>
</mashup>
```

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.

Note: When you override struts that use custom strut classes, see the appropriate Javadocs for other details about your implementation.

Building and Deploying Extensions

After You Create Your Extensions

After you are satisfied with all of the extensions you have made to Sterling Store Associate Mobile, make sure that you build and deploy the new extensions. You should re-build and deploy the Application Enterprise Archive (EAR) with all the Java files, resource files, JSP files, custom classes, and so forth, that you created or modified.

Building Other Extensions

To build your custom code extensions (user exits, extended APIs, custom implementations of supplied Java interfaces, and so forth) modifications, generate a JAR file containing these Java files and custom classes. After creating the JAR file, include the new JAR file in the CLASSPATH environment variable by running the install3rdParty.sh (or install3rdParty.cmd on Windows) utility from the <*INSTALL_DIR*>/bin directory. For example:

```
./install3rdParty.sh <vendorName> <vendorVersion> <-j | -l | -p | -r | -d >
<filelist> [-targetJVM DCL | EVERY | NOWHERE | APP | <Custom cfg type> | AGENT]
```

Here, <*vendorName*> refers to the name of the vendor such as WebLogic, WebSphere, and JBoss. <*vendorVersion*> refers to the version of the vendor. Pass the appropriate argument based on the file type. You can pass the following arguments:

- -j for JAR or compressed files
- -l for shared libraries
- -p for properties files
- -r for resource properties files
- -d for database JAR or compressed files

<*filelist*> refers to the path to your custom file.

Note: If you want to make this custom JAR available to the application server and agents when running the install3rdParty utility, based on your requirement pass the following target JVM arguments:

- DCL—If you want to add the custom JAR to the main dynamic classpath.cfg file only.

- EVERY—If you want to add the custom JAR to all the dynamic classpath files (for example, APPDynamicclasspath.cfg, AGENTDynamicclasspath.cfg , OPSDynamicclasspath.cfg, and ACTIVEMQDynamicclasspath.cfgfiles).
- NOWHERE—If you just want to add the custom JAR to the *<INSTALL_DIR>*/jar directory and do not want to update any of the dynamic classpath files.
- AGENT—If you want to add the custom JAR to the SSADynamicclasspath.cfg file.
- APP—If you want to add the custom JAR to the EAR file. For example, SSA.
- <Custom cfg type> - If you want to add the custom jar to the `<custom cfg type>Dynamicclasspath.cfg` file

For example, if you want to add the custom JAR to the SSADynamicclasspath.cfg file, run the install3rdparty command with following arguments:

```
./install3rdParty.sh <vendor_name_for_custom_jar> <vendor_version_for_custom_jsr>
-j <path_to_custom_JAR> -targetJVM SSA
```

Note: At times, mechanisms supplied by Sterling Store Associate Mobile, such as user exits, are replaced by improved mechanisms. When these mechanism are replaced, they are designated as “deprecated.” Whenever possible, use the new mechanisms rather than the deprecated ones. If you do need to use a deprecated mechanism, it must be run in backward compatibility mode. In addition, note that deprecated mechanisms are supported for two major software versions, and then they are removed from the product.

Deploying Extensions

After you build the required Sterling Store Associate Mobile extensions, you must deploy them.

To deploy the extensions, re-build the EAR file as you did during installation.

IBM recommends that you re-build and deploy the EAR file on your development system and test there first. Then, deploy your extensions to your production system and test them again.

Also, before deploying your extensions on a production system, verify that the *<INSTALL_DIR>*/properties/customer_overrides.properties file has the correct settings. For additional information about overriding properties using the `customer_overrides.properties` file, see the *Sterling Store Associate Mobile: Properties Guide.*

Customizing Web.xml

Customizing web.xml for Multiple Applications

Sterling Store Associate Mobile provides the following extensions for web.xml.

The extension package should have a ".extn" suffix and the "package_name" attribute should be used to specify the package. For example:

```
<WebComponents Package = "package_name.extn">
```

Applications have the capability to suppress some common configurations. Applications can suppress these configurations by using the Suppress element. The suppression will be done by removing any element that matches the suppression criteria. For example:

```
<WebComponents Package = "package_name.extn">
<Suppress>
<servlet><servlet-name>JasperPDFReport</servlet-name></servlet>
servlet-mapping><servlet-name>JasperPDFReport</servlet-name>
</servlet-mapping>
</Suppress>
<web-app>
<!-- All the web.xml pieces needed, in standard web.xml format. -->
</web-app>
</WebComponents>
```

In this example, all configurations are suppressed in which the servlet element contains a child servlet-name with the given name.

Setting Up Backend Logging in the web.xml File

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.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Trademarks