Selling and Fulfillment Foundation

Localizing the Web UI Framework

Release 9.0

March 2010

Sterling Commerce
An IBM Company

Contents

Localization

Web UI Framework Localization

The Web UI Framework allows you to localize the application to handle multiple languages and cultural conventions. You can localize graphical text, messages, units of measurement, and other items.

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

- The default implementation of the Web UI Framework. This implementation provides a default implementation of localization. You can customize this default implementation.

  This option enables you to continue using the localization implementation that you used with previous versions of the application.
- A customized implementation of localization. You can register a customized implementation of localization using the web.xml file.

The use of the same interface contracts by both options ensures a consistent implementation of localization by all users.

Localization is implemented on both the client and server sides of the application. You can localize both text and images. All UI nonfield text and labels are localized on the client side. Field text and images are localized on the server side so that the correct files are referenced by the UI.

After all localization changes are finished, rebuild the EAR/WAR file as you did during the installation, and then deploy the application on the application server.

Interface Contracts of the Web UI Framework - Localization

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

Interface Contract	Description	Methods
LocalizationProvider	Implements ISCUILocalizationProvider, which defines the behavior expected in any implementation of the localization in the application.	• getString(SCUIContext, string message) method Takes the string to be localized and returns the localized string.

Interface Contract	Description	Methods
	LocalizationProvider is plugged in to the application using the context parameter in web.xml: • <param-name> scui-localization-provider • <param-value> com.application.ApplicationLocalizationProvider	When this method is used, the localization service layer (the SCUILocalizationHelper class) uses the SCUIContext value to instantiate the registered LocalizationProvider. • getDBString(SCUIContext, string message) method Takes the string to be localized and returns the localized string from the database. If you are not using this method, use the getString method. When this method is used, the localization service layer (the SCUILocalizationHelper class) uses the SCUIContext value to instantiate the registered LocalizationProvider. • getBundleFor(String name) Takes the form name and returns the name of the bundle. • registerBundle(String bundleName) Registers the localization bundle used by the application. • init Handles initialization.

Web UI Framework Localization - Localizing on the Server Side

Localizing on the server side involves the localization of field text and images that are referenced by the client side. Localizing on the client side involves user interface labels and other nonfield text.

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

1. Use tag libraries to specify the localized text and images that will be referenced from the server side.
2. Register both default and custom localization providers in the web.xml file. Localization uses the LocalizationProvider class to implement the interface class ISCUILocalizationProvider. This interface class defines the basic standards for all localization behavior.
3. Specify the localization provider in an instance of the interface class ISCUILocalizationProvider.

   The following shows an example of ISCUILocalizationProvider:

```
package com.sterlingcommerce.ui.localization;
public interface ISCUILocalizationProvider
{
   public void init();
   public String getString(SCUIContext context, String message);
   public String getDBString(SCUIContext context, String message);
   public String getBundleFor(String name);
```

```
   public void registerBundle(String bundleName);
}
```

To fetch localized strings from a table using the getDBString method, the calling JSP must include the following taglib:

```
<%@ taglib uri="/WEB-INF/scui.tld" prefix="scuitag" %>
```

This taglib is then called in the JSP as follows:

```
<scuitag:i18nDB>test_string</ scuitag:i18nDB>
```

Also, for the getDBString method to fetch localized strings from the table, the dblocalizable property in the entity XML files must be set to **true**.

To fetch localized strings from resource bundles using the getString method, the calling JSP must include the following taglib:

```
<%@ taglib uri="/WEB-INF/scui.tld" prefix="scuitag" %>
```

This taglib is then called in the JSP as follows:

```
<scuitag:i18n>test_string</ scuitag:i18n>
```

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 Localization - Localizing on the Client Side

Localizing on the client side involves the localization of user interface labels and other UI non-field text and images. Localizing on the server side involves the localization of field text and images that are referenced by the client side.

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

1. To localize images, set up a locale-specific CSS file that will be used by a tag library.
2. To localize text, set up a global JavaScript bundle file. This bundle file includes pairs of key values (the string to be localized and the localized value of that string). The Web UI Framework provides a tag library that includes the global bundle file.

   Global bundle file example:

   Note: Sterling Commerce recommends that you prefix bundle file keys with "b_" to avoid conflicts with other properties on the screen.

```
Ext.override(sc.plat.ui.Screen, {
    'b_key1': 'value one',
    'b_key2': 'value two',
    'b_key3': 'value three'
})
```

   Each form in the application has a specific bundle.js file that should be included with the screen file. For example, if the bundle file of form login.js is sc.ui.login.bundle.js, when a screen is accessed, the tag library looks for that bundle file and form and includes them with the screen. The tag library does not automatically include the bundle file for a screen. For a localizable file (such as a bundle file) that is included using the

scuitag command, the tag automatically includes the localized file (fr, de, jp, etc.), according to the user's locale.

When an individual file using JSB definitions is included, the entry for the bundle file should be made after the entry to include the screen file. The screen files should be included in the following order:

1. <screen>_config.js
2. <screen>.js
3. <screen>_bundle.js

Example of a screen referencing a bundle file:

```
com.xyz.MyScreen = Ext.extend(sc.plat.ui.ExtensibleScreen,
{
        className: "sc.plat.ui.Screen",
        getUIConfig: function() {
                return {
                        title: this.b_key1,
                        items:[{
                                xtype: 'label',
                                text: this.b_key2
                        }]
                 }
        },
        getInfoMessage: function() {
                return this.b_key3;
        }
});
```

The following shows an example of how a form is set up to return a bundle file name when the getBundleFor(form name) method of the SCUILocalizationProvider class is called by the screen. The keyword **this** returns the instance of the form, so this.First_Name returns the bundle file **First Name**.

```
First_Name = "First Name";
Last_Name = "Last Name";
Search_View = "Search View";
Ext.onReady(function(){
var form = new Ext.FormPanel({
        name: 'view',
        title: this.Search_View,
        xtype: 'form',
        items: [{
                       fieldLabel: this.First_Name,
                       xtype: 'textfield',
                       appValidator: validatorApplication,
                       validationEvent: 'blur',
                       validator: validateString
                 },
                 {
                       fieldLabel: this.Last_Name,
                       xtype: 'textfield',
                  }]
         });
         form.render(Ext.getBody());
});
```

Web UI Framework Localization - Setting Locale-Specific Formats

You can set the following locale-specific formats in a JavaScript file:

- Date/time
- Decimal
- E-mail
- Phone number
- Credit card

The validation.js file is a localizable file. This file is available at <install>/repository/eardata/platform_uifwk/<version>/war/platform/scripts/. In the directory path <install>/repository/eardata/platform_uifwk/<version>/war/, <version> is either **20** or **30** depending on if the application is using Ext JS 2 or Ext JS 3.

You can copy this validation.js file into <install>/repository/eardata/platform_uifwk/<version>/war/localization/<locale_directories>/platform/scripts/ and modify it as needed.

This validation.js file is given priority over the validation.js file in the <install>/repository/eardata/platform_uifwk/<version>/war/platform/scripts/directory.

After the localization is finished, the JavaScript file must be regenerated and minified. For more information, refer to the information on compiling and minifying JavaScript files.

Date/Time Formats

Date and time formats can vary among countries. For example, April 2, 2003 could be written as 02/04/03 in one country and 04/02/03 in another country. You can use the validation.js file to localize date/time formats.

In validation.js, all date/time formats are in the PHP format. The validation.js file includes a Java-to-PHP conversion table.

User date/time formats are set as follows:

```
sc.plat.Userprefs.setDateFormat('m/d/Y');
sc.plat.Userprefs.setTimeFormat('H:i:s');
sc.plat.Userprefs.setTimestampFormat('Y-m-d\\TH:i:sP');
sc.plat.Userprefs.setMonthDisplayDateFormat('F Y');
sc.plat.Userprefs.setDayDisplayDateFomrat('N j');
sc.plat.Userprefs.setHourMinuteTimeFormat('H:i');
sc.plat.Userprefs.setDateHourMinuteFormat('m/d/Y H:i');
```

Server date/time formats are set as follows:

```
sc.plat.info.Application.setDateFormat('Y-m-d');
sc.plat.info.Application.setTimeFormat('H:i:s');
sc.plat.info.Application.setTimestampFormat('Y-m-d\\TH:i:sP');
```

The validation.js file also defines the following methods for converting the date/time between the server and user formats. In a localized file these methods can be modified.

- sc.plat.DateFormatter.dateFormatConverter

  Converts the date between the server and user formats.

- sc.plat.DateFormatter.timeFormatConverter

Converts the time between the server and user formats.

Decimal Formats

The decimal format is set as follows:

```
sc.plat.Userprefs.setDecimalSeparator(".");
```

For example, in a German locale, this format would be set to:

```
sc.plat.Userprefs.setDecimalSeparator(",");
```

E-mail Addresses, Phone Number, and Credit Card Formats

The formats for e-mail addresses, phone numbers, and credit cards are set as follows. In a localized file, you can register new validators that are specific to a locale.

- `sc.plat.ValidateUtils.registerValidator ('email',sc.plat.ValidateUtils.emailFormatValidator);`
- `sc.plat.ValidateUtils.registerValidator ('creditcard',sc.plat.ValidateUtils.creditCardFormatValidator);`
- `sc.plat.ValidateUtils.registerValidator ('phone',sc.plat.ValidateUtils.phoneNumberFormatValidator);`

Web UI Framework Localization - Localizing the Structure of Panel Components

You can use the Web UI Framework to localize the structure (or order) of components in a screen panel, using a locale-specific bundle file. You can re-arrange the order of both words and graphics in a sentence.

Use sentence-based panels to localize the structure of panel components. The Web UI Framework lets you use static text for labels and editable controls that can be moved.

The following properties are used in the Ext.Panel object:

- scLocalizedKey

  The string that will be used as the basis for which controls will be created and/or re-arranged.

- scLocalizedLabelCss

  The localized label class that is applied to the label. The Web UI Framework provides this class.

- sc-plat-localizedLbl

  The class that is applied to all labels created by the Web UI Framework in a localized panel.

Web UI Framework Localization - Example of Localizing the Structure of Panel Components

The following examples shows two different outputs from the following sentence-based localized panel:

```
Ext.panel instance
      |
      |__scLocalizedSKey = "DiscountAmount"
      |
```

```
      |__scLocalizedLabelCss= "custom-css"
      |
      |__items (children)
           |
           |__Amount TextField
           |
           |__Discount comboField
```

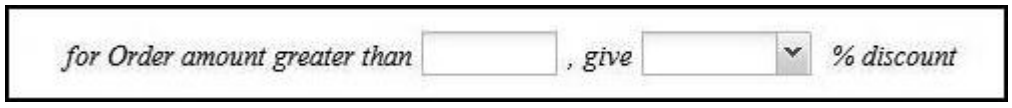
Using this instance of Ext.panel, the panel could appear differently in two locales. The %sciId values refer to the controls (text field and combo field).

Locale A code:

```
DiscountAmount = " for Order amount greater than {%sciId0} , give {%sciId1} %
discount."
Ext.panel instance

    |
    |__children
          |
          |__label "for Order amount greater than "
          |
          |__Amount TextField
          |
          |__label ", give "
          |
          |__Discount comboField
          |
          |__label " % discount. "
//all labels created with style 'custom-css' and 'sc-plat-localizedLbl'
```

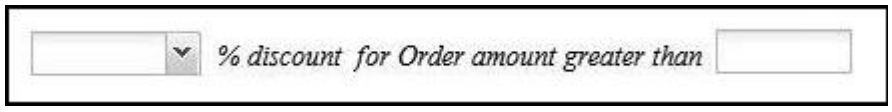
Locale A output:



Locale B code:

```
DiscountAmount = "{%sciId1} {xtype:'label,text:'% discount, for Order
amount greater than',cls:'custom-css'} {%sciId0} ."
Ext.panel instance
      |
      |__children
            |
            |__Discount comboField
            |
          |__string ``{xtype:'label,text:'% discount, for Order
amount greater than',cls:'custom-css'}`` will be passed to eval.
            |
            |__Amount TextField
```

Locale B output:



Web UI Framework Localization - Localizing UI Branding

The Web UI Framework allows you to customize UI branding in two ways.

- You can customize the Look and Feel of the UI by means of CSS files.
- You can customize the UI layout by using the Web UI Framework Extensibility Workbench.

Localizing Themes

The Look and Feel (which include themes and images) of the UI in the Web UI Framework is CSS file driven. You can add or override or modify these CSS files. All of the default theme CSS files are stored in the << *Web_Context_Root*> >/platform/css directory. You can add your own theme-specific CSS files in the < *Web_Context_Root*>/<*locale*>/<*application_id*>/css/ directory.

If you are overriding a CSS file, you need to add the overridden entries in the localized directory (< *Web_Context_Root*>/<*locale*>/platform/css/platform.css).

You can override the default CSS (< *Web_Context_Root*> /platform/css/platform.css) by adding the overridden entries to the platform.css file and copying the overridden file to the localized directory (< *Web_Context_Root*> /<locale>/platform/css/platform.css)

Localizing Icons

You can add your new images and/or icons to the < *Web_Context_Root*> /<*custom_path*> directory. Also, add an entry for these images and/or icons in your localized CSS file.

If you overriding an existing image and/or icons, you need to add the overridden image and/or icons to the localized directory (< *Web_Context_Root*>/<*locale*>/platform/icons). Also, add an entry for these images and/or icons in the localized CSS file.

Customizing UI Layout

You can customize the UI layout by using the Web UI Framework Extensibility Workbench. For more information about the Web UI Framework Extensibility Workbench, refer to Customizing the Web UI Framework Guide.

Web UI Framework Localization - Bundle Collector Utility

On the client side, you can index localization bundle files from different directories of the application using the JavaScript bundle collector and collator utility.

This tool does the following:

- Collects, into an index file, bundles defined in multiple JavaScript files (ending with _bundle.js).

  The index file's format is the standard Java properties file format. Duplicate bundle values are not included. Context-sensitive comments are included in the bundle file for reference during localization, only if the bundle value includes characters required for formatting a string and has a preceding comment.

- Generates JavaScript bundle files, given a localized bundle index file.

  Once the bundle index file is localized, the corresponding localized JavaScript file must be regenerated and minified. For more information, refer to the information on compiling and minifying JavaScript files.

The following shows examples of a bundle file and a corresponding index file.

JavaScript bundle file:

```
Ext.override(sc.plat.ui.Screen, {
    b_First_Name: "First Name",
    b_Last_Name: 'Last Name',
    // {1} - last name, {0} - first name (should be picked up by collector)
    b_Name: '{1}, {0}',
    // a context insensitive comment (should be ignored by collector)
    b_Search_View: "Search View"
});
```

Index file:

```
1:First Name
2:Last Name
3:Search View
#// {1} - last name, {0} - first name (should be picked up by collector)
4:{1}, {0}
```

The bundle utility uses the following utility class:

com.sterlingcommerce.ui.web.framework.build.utils.SCJSBundleUtil

This class can run in the following modes, depending on what task you want to perform:

- index

  Generates an index of JavaScript bundle files.

- map

  Generates localized JavaScript source files from the localized bundle index file.

The following tables show the arguments for the two modes:

index

Arguments	Example

• -index
The mode name.
• -sourcedir
The JavaScript source directory. It could be your webcontent directory or any of its parent directories.
• -indexdir
The directory to contain the generated index metadata. It is required when you are mapping the bundle back to the JavaScript source files.
• -webcontentdepth
(Optional) The depth of the webcontent directory. This defaults to 0.
• -ignorefilter
(Optional) Indicates what folders to exclude from the index. This defaults to `.*/localization/.*`
	`-index`
	`-Dsourcedir=<install>/repository/eardata/<app_dir>/war`
	`-Dindexdir=<install>/repository/eardata/<app_dir>/localization_index`
	`-webcontentdepth=1`
	`-debug`

Arguments	Example
• -debug (Optional) Indicates whether to run with the debug log enabled. This defaults to N (do not run with debug log enabled).	

map

Arguments	Example

• -map
The mode name.
• -sourcedir
The JavaScript source directory.
• -indexdir
The original index metadata directory which was localized.
• -indexfiles
One or more localized index files. For example, a localized file could be named bundle-index_en_US.properties.
• -localizationdir
(Optional) The localization output directory where the localized JavaScript source files will be generated. If it is not provided, then the <webcontent directory>/localization directory will be used.
The index file will be generated at <outputdir>/ localization/ <LOCALE_DIRECTORIES>/ <originalfilepath>
LOCALE_DIRECTORIES refers to the locale. For example, for locale fr_FR, it is **/fr/FR**.
• -webcontentdepth
(Optional) The depth of the webcontent directory. This defaults to **0**. The localized files will be generated in the webcontent directory.
• -ignorefilter
(Optional) Indicates what folders to exclude from the index. This defaults to `.*/localization/.*`
• -debug
(Optional) Indicates whether to run with the debug log enabled. This defaults to N (do not run with debug log enabled).
	`-map`
	`-Dsourcedir=<install>/repository/ eardata/<app_dir>/war`
	`-Dindexdir= <install>/repository/ eardata/<app_dir>/localization_index`
	`-Dindexfiles=/INSTALL_TEST/ Platform/fairlop/ranadive/RC3/ install/repository/eardata/appdir/ localization_index/bundle-index_fr_FR.properties`
	`-webcontentdepth=1`
	`-debug`

You also can use an ant XML utility (jsUtil.xml) to invoke the utility class com.sterlingcommerce.ui.web.framework.build.utils.SCJSBundleUtil. This ant utility can be used to test and run the utility classes. This utility does not expose everything that is supported by the class, but the supported

items that it does expose can be used in most cases. If you need to invoke the actual class with custom arguments, this XML file can be used as a sample source file.

The jsUtil.xml file is located in the <*INSTALL_DIR*>/bin folder after the Web UI Framework is installed. Sample code for invoking the ant utility in the two different modes is shown below:

• index mode

```
./sci_ant.sh -f jsUtil.xml bundle.index
-Dsourcedir=<INSTALL_DIR>/repository/eardata/<app_dir>/war
-Dindexdir=<INSTALL_DIR>/repository/eardata/<app_dir>/localization_index
```

• map mode

```
./sci_ant.sh -f jsUtil.xml bundle.map
-Dsourcedir=<INSTALL_DIR>/repository/eardata/<app_dir>/war
-Dindexdir=<INSTALL_DIR>/repository/eardata/<app_dir>/localization_index
-Dindexfile=<INSTALL_DIR>/repository/eardata/<app_dir>/localization_index/bundle-index_en_US.properties
```

Running the Bundle Collector Utility to Localize Platform Files

You can also use the bundle collector utility to generate index and bundle files for some platform_uifwk files like the dashboard bundle files.

The procedure is the same as above, except the directory structure would change to the following:

In <install>/repository/eardata/platform_uifwk/<version>/war/<directory containing required files>, <version> is either **20** or **30**, depending on if the application is using Ext JS 2 or Ext JS 3.

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>
```

Web UI Framework Localization - Final Localization Tasks

After all localization changes are finished, rebuild the EAR/WAR file as you did during the installation, and then deploy the application on the application server.