MOTODEV

The Motorola developer network

DEVELOPING AN INTERNATIONALIZED WEBUI APPLICATION

V 1.0

TECHNICAL ARTICLE

Developing an Internationalized WebUI Application

Version 1.0

December 2008

For the latest version of this document, visit http://developer.motorola.com

Motorola, Inc. http://www.motorola.com

Contents

Introduction

The purpose of this article is to demonstrate a simple Planner application which retrieves information from the Mobile Calendar using the APIs exposed by WebUI application environment. The application demonstrates the use of internationalization. The Integrated Development Environment (IDE) used for development of this application is MOTODEV Studio for WebUI.

This article discusses the development of a WebUI application which does the following:

- Access platform APIs exposed by the WebUI application environment
- Access external web services (demonstrate usage of APIs on the internet)

By combining the above two functionalities, a developer can develop appealing applications using WebUI.

The application is a simple Planner application which retrieves information from the built in Calendar on the mobile device. The application also provides the additional feature of being internationalized for multiple languages.

The demo application accesses platform APIs like Calendar, Date, i18n, etc., and displays the information via the user interface.

An external web service is required to perform the language translation of user text.

The data to be presented via the example UI is as shown below:

If the above is the UI data to be presented for a multi language application, it can normally handle the translation of headings such as "Time" and "Event" in the code using a lookup to reflect correctly in multiple languages. However, it is more difficult to handle the semantic correctness of “Fill in Petrol on way back home” in another language.

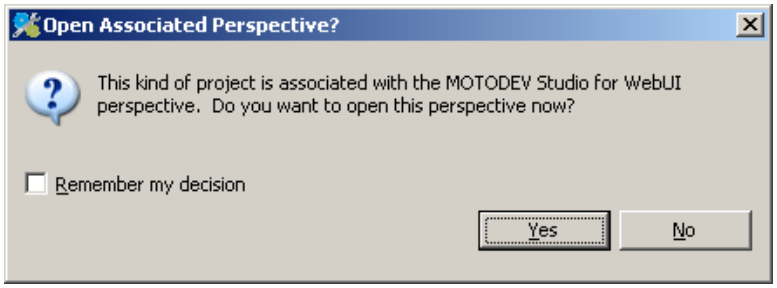
With WebUI, using an external language conversion Web Service, the developer can handle the user entered text (“Fill in Petrol on way back home”) dynamically for a better translation equivalent in the language the application end user chooses.

The application would also require navigation capabilities to check more events existing in the Calendar on the mobile device.

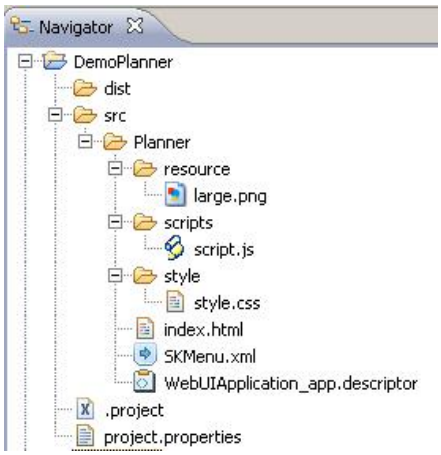
The article introduces the development in phases. The first phase covers the development of the application for a single language. The second phase covers internationalization of the application. The third phase covers the use of an external web service for achieving dynamic translation to provide better translation (semantic meaningfulness) for the application.

Phase1: Initial application development

Open MOTODEV Studio for WebUI and create a new WebUI Application Project. This can be done by clicking on File > New > WebUI Application Project. This presents a screen requesting the Project name, workspace location and WDK options. Populate the fields presented and select the checkbox ‘Blank Application’ under ‘MOTOMAGX WDK’. This step requests a ‘Blank Application Name’ to be filled in. Provide the name ‘Planner’. After clicking finish, the project is created.



The above dialog box appears the very first time you create the project using MOTODEV Studio for WebUI. Select the checkbox – “Remember my decision” and click on the button ‘Yes’.



Directory structure of a WebUI application project

The generated project has two main folders ‘dist’ and ‘src’. The folder ‘src’ has the subfolder ‘Planner’ which contains the main source code for the application. The other files ‘.project’ and ‘project.properties’ have information regarding the project.

The ‘Planner’ folder contains the following:

- ‘index.html’ - the main start up file containing the UI presentation
- ‘SKMenu.xml’ - a configuration file for defining SoftKeys
- ‘WebUIApplication_app.descriptor’ - the metadata for the Planner application
- ‘resource’ subfolder - for logically grouping all the resource files associated with the project

- ‘scripts’ subfolder - to logically group all the JavaScript files used for the WebUI application
- ‘style’ subfolder - to logically group all the style sheets associated with the application project

The coding for the application follows the Model View Controller (MVC) design pattern.

The ‘Model’ component contains a JavaScript file which provides the interface to the platform API and also any web service request and response handling. This requires creating a new JavaScript file ‘scriptEngine.js’ This can be done by right clicking on the scripts folder, New > File. When prompted, name it ‘scriptEngine.js’.

The ‘View’ component contains:

- the ‘index.html’ file,
- any number of CSS files (style folder) to add to the aesthetic appeal of the UI, and
- any number of images (res folder) to be used for display

The ‘Controller’ component is comprised of a JavaScript file ‘script.js’. This has all the control logic of the application including the configuration settings for the soft keys.

View Component

For Display Handling (html file):

1 Double click index.html to open it.

2 Add the following Code (Containers) to handle Date:

```
<div id="dateDisplay" style="display:block">
<h4 id="dateConst">DATE</h4>
<p id="dateFormat"></p>
</div>
```

- The **Div** tag with id "dateDisplay" is a self contained display area to display Date in the html code.
- The **h4** tag with id "dateConst" is a header tag to display the header "DATE".
- The **p** tag with id "dateFormat" is a tag to display the actual date in appropriate format DD/MM/YYYY dynamically populated by JavaScript code.

3 Similarly add the following Code to handle Time:

```
<div id="timeDisplay" style="display:block">
<h4 id="timeConst">TIME</h4>
<p id="timeFormat"></p>
</div>
```

- The **Div** tag with id "timeDisplay" is a self contained display area to display Time in the html code.
- The **h4** tag with id "timeConst" is a header tag to display the header "TIME".
- The **p** tag with id "timeFormat" is a tag to display the actual start time and end time in appropriate format HH:MM (24 Hour format) dynamically populated by JavaScript code.

4 Similarly add the following Code to handle Event:

```
<div id="eventDisplay" style="display:block">
<h4 id="eventConst">EVENT</h4>
<p id="eventFormat"></p>
</div>
```

- The **Div** tag with id "eventDisplay" is a self contained display area to display the Event in the html code.
- The **h4** tag with id "eventConst" is a header tag to display the header "EVENT".
- The **p** tag with id "eventFormat" is a tag to display the actual event stored in mobile calendar dynamically populated by JavaScript code.

For Display Styling (CSS file):

CSS file defines the rules on how to display the tags defined in the above html.

1 Double click ‘style.css’ to open it.

2 Enter the following lines of code:

```
#dateConst
{
      text-align:left;
}
#timeConst
{
      text-align:left;
}
#eventConst
{
      text-align:left;
}
```

The above code sets the text alignment of text to be displayed as left-justified for all the text under the tags with these IDs `dateConst, timeConst, eventConst`.

3 Enter the following lines of code:

```
#dateFormat
{
      text-align:center;
}
#timeFormat
{
      text-align:center;
}
#eventFormat
{
      text-align:center;
}
```

The above code sets the text alignment of text to be displayed as centre justified for all the text under the tags with id's dateFormat, timeFormat, eventFormat.

For background colour, type the below code:

```
body
{
      background-color: #0f0faf;
}
```

Model Component

The Model Component ('scriptEngine.js') interfaces with the internal WebUI application environment APIs and external Web Services. For Specific API details via MOTODEV Studio for WebUI, Go to Status bar click on Help > Help Contents > MOTODEV Studio for WebUI User Guide > Reference > API Reference.

For the application, the following details are required:

- List of Events
- Start Date of Event
- Start Time and End Time of Event
- Event Summary

To obtain a List of Events, create a method `getEventsList()`. This method queries the WebUI application environment's Calendar Object, `getEvents()`. This platform method, `getEvents()`, takes two arguments, `starttime` and `endtime`, which must be supplied. The date objects to be supplied can be obtained by querying the current time. The WebUI application environment provides the method, `getCurrentTime()`, which returns an instance of Date Object similar to the standard W3C Date Object. The following tasks are performed in the `getEventsList` method:

1. Obtain the current time by using the code snippet below:

   ```
   var currentDate = webui.system.util.time.getCurrentTime();
   ```

2. Manipulate the instance of Date object for a future date (7 days) by using code snippet below:

   ```
   var futureDate = webui.system.util.time.getCurrentTime();
   futureDate.setDate(futureDate.getDate()+7);
   ```

3. The platform method `getEvents` can now be invoked as below:

   ```
   var eventsList = webui.calendar.getEvents(currentDate,
   futureDate);
   ```

The `eventsList` thus obtained contains a list of all the mobile calendar events which fall within a week timeframe.

For `StartDate`, create a method `getEventStartDate()`. This method makes use of the `eventsList` obtained earlier and gets the start date of the event. The following is the code snippet to use where `eventIndex` is the index of the event in the obtained `eventsList`.

```
eventsList[eventIndex].getStart();
```

To obtain the individual day, create a method `getEventStartDate()` which performs the internal operation as below:

```
eventsList[eventIndex].getStart().getDate();
```

To obtain individual month, create a method `getEventStartMonth()` which performs the internal operation as below:

```
((eventsList[eventIndex].getStart().getMonth())+1);
```

The addition of the number '1' is required as Month is indexed from 0. For example, January corresponds to 0 and December corresponds to 11 in Javascript.

To obtain the individual year, create a method `getEventStartYear()` which performs the internal operation as below:

```
eventsList[eventIndex].getStart().getFullYear();
```

To obtain individual hour, create a method `getEventStartHour()` which performs the internal operation as below:

```
eventsList[eventIndex].getStart().getHours();
```

To obtain individual minutes, create a method `getEventStartMinute()` which performs the internal operation as below:

```
eventsList[eventIndex].getStart().getMinutes();
```

To obtain the end time details of the event, the following code snipped can be used:

```
eventsList[eventIndex].getEnd();
```

Similarly Events end Hour can be obtained by creating a method `getEventEndHour()`:

```
eventsList[eventIndex].getEnd().getHours();
```

Similarly Events end Hour can be obtained by created a method `getEventEndHour()`:

```
eventsList[eventIndex].getEnd().getMinutes();
```

To obtain the event summary, create a method `getEventSummary()` which performs the internal operation as below:

```
eventsList[eventIndex].summary;
```

Controller Component

The Controller Component controls what gets displayed at what time. All the Navigation handling is controlled. The code goes into the 'script.js' file.

The `onLoadAction()` method has control over what is displayed on the UI when the application is first launched. On launch, the first event should be displayed if available, else a screen with "No Events" should be displayed. For navigation purposes, soft keys must be displayed on the User Interface.

Soft Keys

Soft Keys are used for navigation and controlling the UI of the application. Soft keys are the top two buttons on the keypad of a Motorola phone which aid in easy interacting with device. The navigation in the application being developed requires Transitioning from one event to another and exiting the application. Hence the possible transitions are Next, Previous and Exit. Different screen contexts would have different soft key menus. To provide soft key menus, two methods can be followed – either the Graphical editor of 'SKMenu.xml' or the text based editor of 'SKMenu.xml' file.

For a UI when navigation requires 'Next' and 'Exit', then the following has to be done:

```
<ScreenContext id="Planner">
	<item>
		<name>Next</name>
		<action>getNext</action>
	</item>
	<item>
		<name>Exit</name>
		<action>goExit</action>
	</item>
</ScreenContext>
```

Here ‘item’ is the soft key while ‘name’ is the option that is associated with the soft key to appear on the UI. The ‘action’ is what needs to be performed when the soft key is pressed.

For a UI when no events exist, then only one soft key is required, namely exit. This can be done as follows:

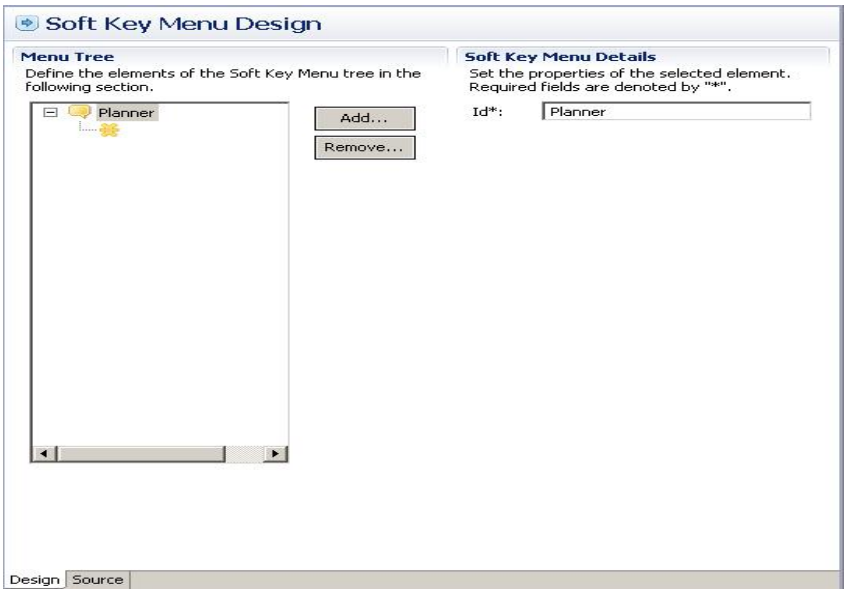
```
<ScreenContext id="Planner4">
	<item>
		<name/>
		<action/>
	</item>
	<item>
		<name>Exit</name>
		<action>goExit</action>
	</item>
</ScreenContext>
```

Similarly for a UI that needs all actions (Next, Previous and Exit) the following has to be added:

```
<ScreenContext id="Planner2">
		<item>
			<name>Options</name>
			<action/>
		<item>
			<name>Next</name>
			<action>getNext</action>
		</item>
		<item>
			<name>Previous</name>
			<action>getPrevious</action>
		</item>
		</item>
		<item>
			<name>Exit</name>
			<action>goExit</action>
		</item>
</ScreenContext>
```

The left Soft Key shows Options and when pressed, opens a sub menu showing the two items.

The Graphical View of the ‘SKMenu.xml’ file helps to easily create the above xml file without the need to remember the xml structure.



Initial definition of screen context

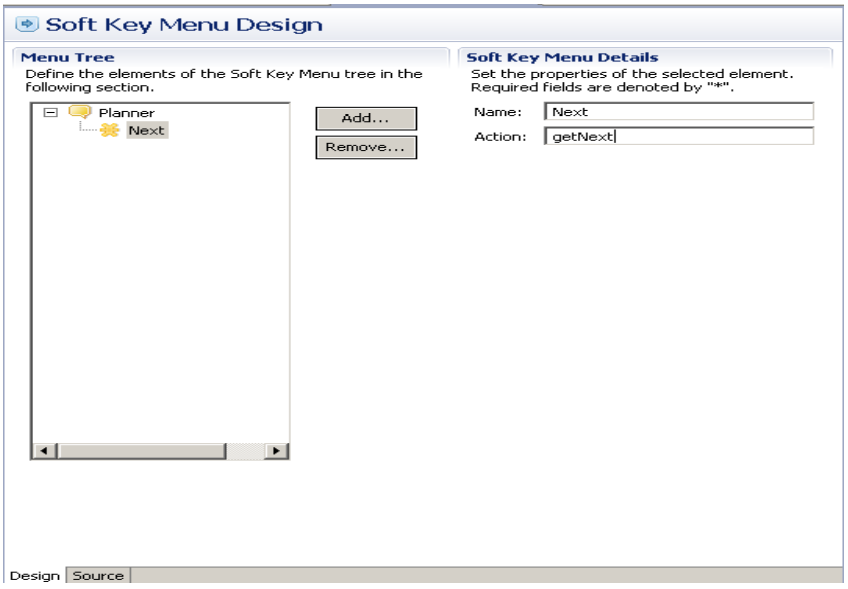
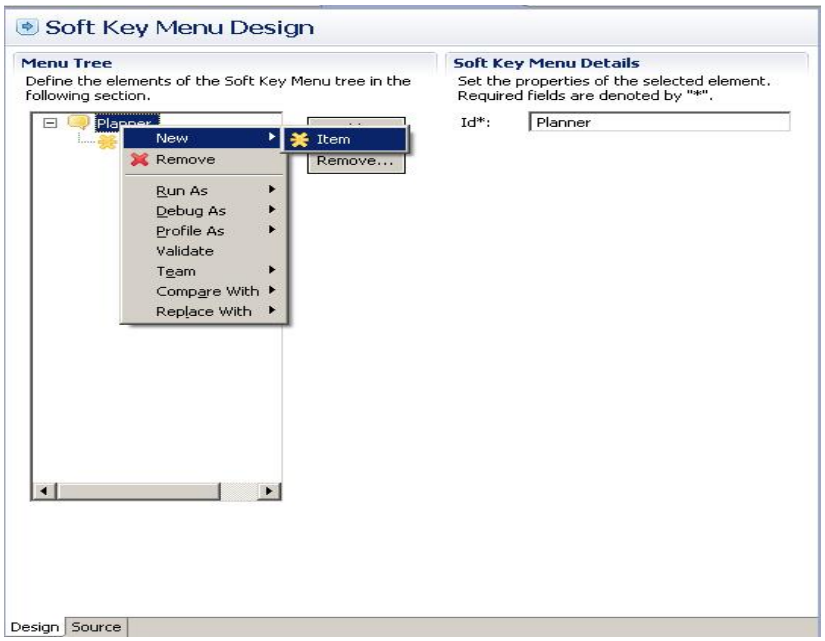


Figure: Definition of Soft Key



Adding more soft keys

Right clicking on the screen content gives more options to add/remove soft keys.

Initializing the Soft Key menu in the 'script.js' file can be done as follows:

```
var  menu = webui.softKeys.loadMenu("Planner");
```

To activate the menu, the following has to be done:

```
webui.softKeys.setActiveMenu(menu);
```

To listen to the Soft Key events, a Listener is needed and a handler to handle those events. The listener can be coded as below:

```
var listener = webui.softKeys.createListener();
```

A handler function must be associated with the listener:

```
listener.handle = handleSoftKeys;
```

The menu must be associated with the listener:

```
menu.addListener(listener);
```

The function `handleSoftKeys()` contains the logic associated with the necessary action to be performed when a soft key is pressed. The skeleton of which is as follows:

```
function handleSoftKeys(action)
{
if (action == 'getNext')
{
//Action to be performed
}
}
```

The control logic decides the contents populated to the UI. The UI is populated using the methods exposed by the Model Component 'ScriptEngine.js'. The html pages are dynamically populated as follows:

```
document.getElementById("dateFormat").innerHTML =
" "+getEventStartDate()+"/"
+getEventStartMonth()+"/"
+getEventStartYear();
```

The HTML Document Object Model (DOM) structure is parsed for the tag with the ID ‘dateFormat’ and the contents are inserted with the necessary formatting done.

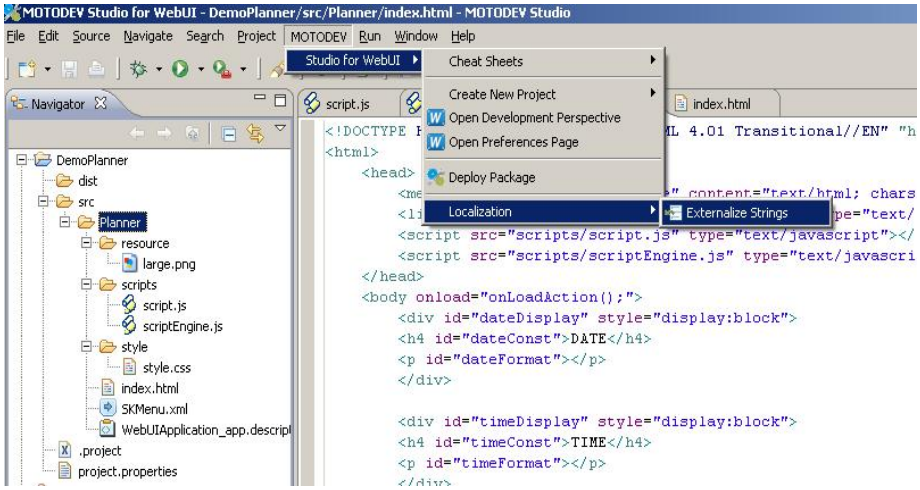
The Code at the end of phase 1 is a full fledged working sample of the simple planner application. The sample files at the end of phase1 are shown in Appendix A.

Phase 2: Internationalization

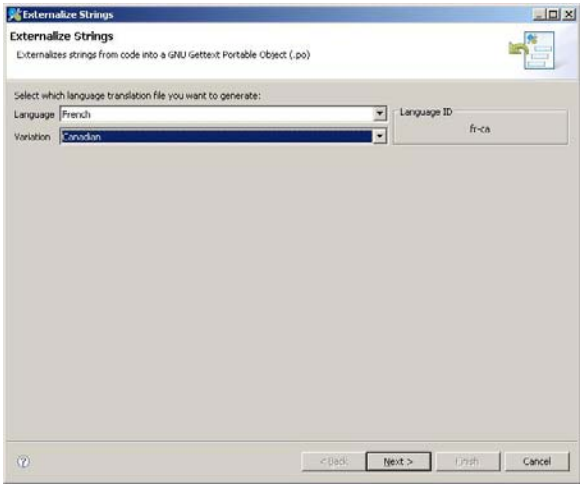
The application designed so far only caters to a select audience of English speaking people. Generally for any application developer, the desire for the application to reach out to the maximum number of users spanning the whole world is a key goal. Reaching out to the global audience in an appeakubg manner means the developer essentially has to understand the key user requirement that “not everyone speaks and understands a single language". Hence the developer must take care to design the application to cater to various languages by employing internationalization. Internationalization is the process of designing a software application so it can be adapted to various languages and regions without additional engineering changes. There are advantages and disadvantages to note for this approach. The disadvantage is internationalization could mean a trade-off on some usability for people in certain localities/cultures/languages. The advantage is a significant reduction in the development and maintenance costs for customized applications.

Internationalizing applications using MOTODEV Studio for WebUI is a simple job. The following tasks need to be done:

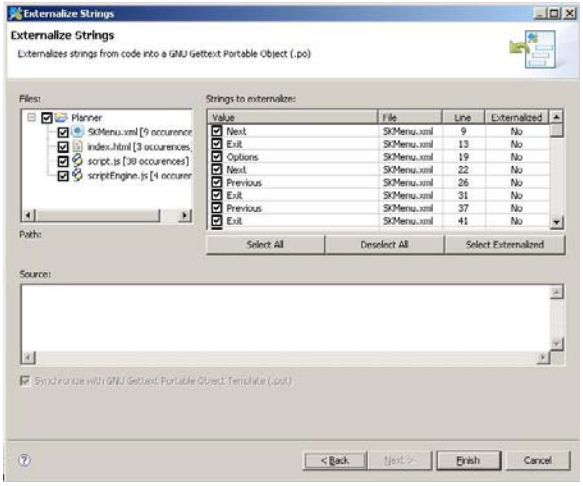
1. Select the folder ‘Planner’ which is under ‘src’ folder of the project.
2. Go to the Status bar and choose MOTODEV > Localization > Externalize Strings



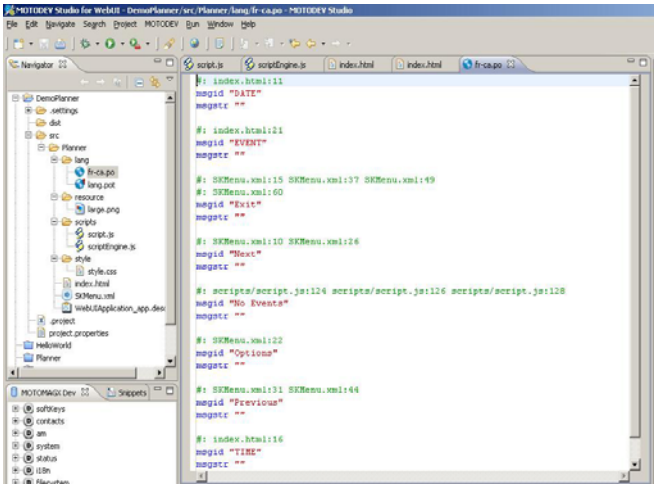
3. The above step presents a new form. Select the language and variation for externalization purpose. For this example, let the language be ‘French’ and the variation be ‘Canadian’ and click ‘Next’



4. The tool presents a form with all the hard coded strings in the project for externalization purposes. Select the appropriate ones and deselect the others (e.g., Debug Print Statements) and click finish.



5. The above generates code replacing all the hard coded strings and also creates a separate ‘lang’ folder with ‘.po’ files containing the externalized strings.

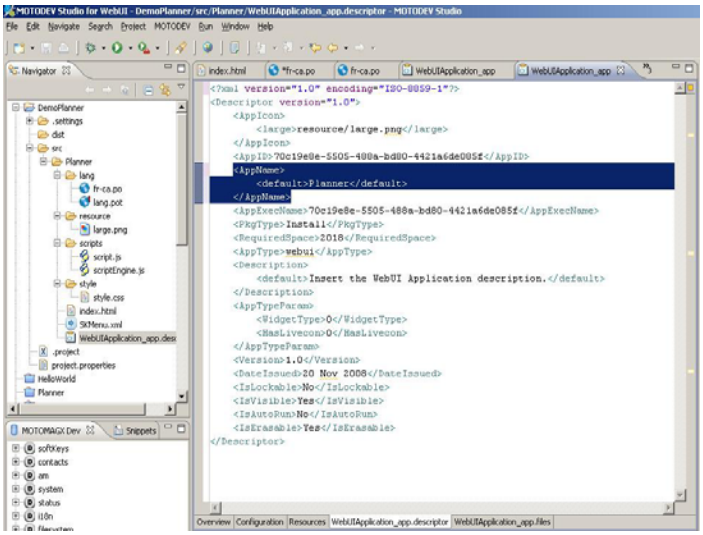


The hard coded strings are replaced by the code as follows:

```
webui.i18n.localize("DATE");
```

The `webui.i18n` string is an object exposed by the WebUI application environment for performing internationalization operations. This object exposes an API, `localize()`, which can be used for localization purposes.

6. Populate the string translations in the msgstr fields as appropriate.
7. For externalizing, the name which appears on the “Planner” menu, Open the “WebUIApplication_app.descriptor” file and go to the text editing mode by clicking on the ‘WebUIApplication_app.descriptor’ tab.



8. Under <AppName> tag in the xml representation add a new tag <ResourceID> and Planner as the value between the nodes. The resulting <AppName> tag is shown:

```
<AppName>
        <default>Planner</default>
        <ResourceID>Planner</ResourceID>
  </AppName>
```

9. Add an additional line in the ‘fr-ca.po’ file to incorporate “Planner” as follows:

```
#: WebUIApplication_app.descriptor:9
msgid "Planner"
msgstr "Planificateur"
```

The above application is now localized to handle the user option of French. Similarly the same steps could be repeated (except Step 8) for other languages.

Phase 3: Utilizing an external web service

The application coded so far handles the language translation for all the hard coded strings. However, the Events have data in the Calendar which are based on the language used when they were entered into the mobile calendar. However, if the user has chosen a different language at a later point in time, then the translation performed could be meaningless. To make the application semantically more appropriate, make use of a language translation web service available for performing the appropriate translation. One such service is offered by Google - Language Translation Ajax API. The following modifications have to be done to the above code to make use of this feature:

1. In 'index.html' file, add the following line along with other script tags:

```
<script type="text/javascript"
src="http://www.google.com/jsapi"></script>
```

2. In the 'script.js' file, perform the language load api:

```
google.load("language", "1",
{
callback:function()
{
      println("Google Language Load Completed");
}
});
```

3. In 'scriptEngine.js' file, perform the following operations to handle language translation:
   - Obtain the language on the device using the call:

```
var currentLanguage;    //Holds the current language set
currentLanguage =
webui.system.util.resources.getCurrentLocale().language;
```

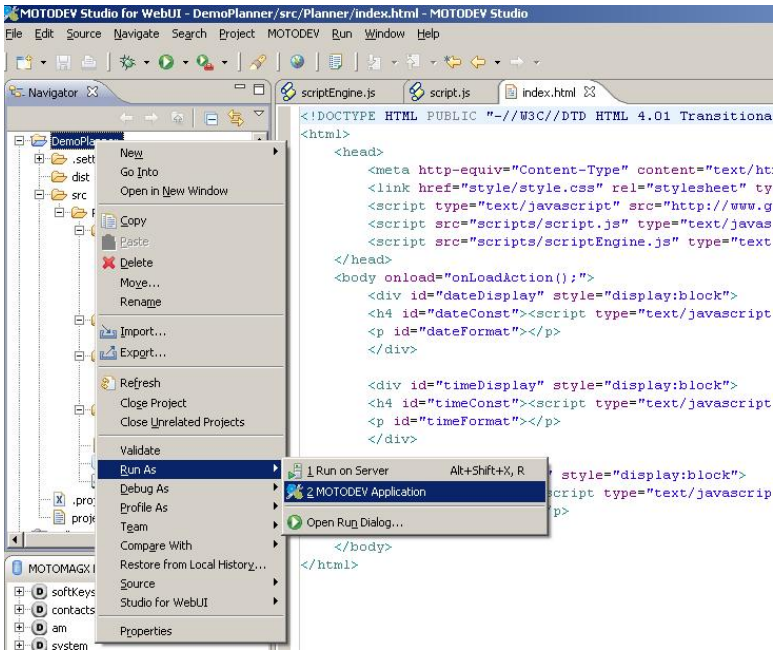
   - Perform the Google translate api call as follows:

```
google.language.translate(summary, '', currentLanguage,
function(result)
{
if (!result.error)
{
translatedSummary = result.translation;
}
});
```

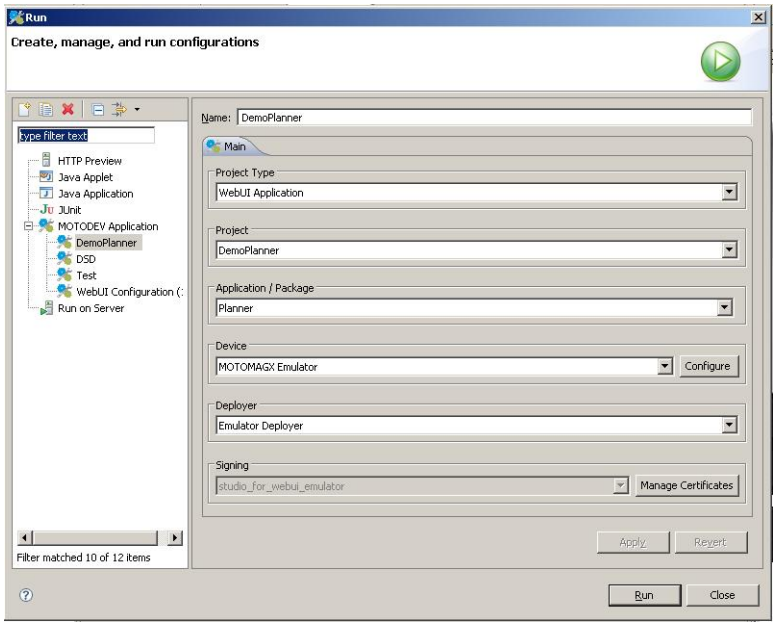
4. The above 'translatedSummary' variable holds the translated text returned by the WebService.

To deploy the application under a specific device menu, it must be specified in the 'WebUIApplication_app.descriptor' file. This can be done on the 'configuration tab' of the file under 'Application Installation Folder' options. Select the folder from the drop down menu option and save the file.

The application can be deployed by right-clicking on the project folder and choosing Run As > MOTODEV Application.



When selected, the following screen is presented:



Give a name to the runtime configuration “DemoPlanner” and click on Apply and then “Run”. This will deploy the application on the emulator.

To test the application, write the following test code to insert some calendar events:

```
var testEvt = webui.calendar.createEvent();
var stdt = webui.system.util.time.getCurrentTime();
stdt.setDate(stdt.getDate()+2);
testEvt.setStart(stdt);
stdt.setHours(stdt.getHours()+1);
testEvt.setEnd(stdt);
```

```
testEvt.summary = "test event one";

webui.calendar.addEvent(testEvt, false);
```

Similarly create a few more events and then run the application to see the results. Below are the screenshots of the application running on the MOTODEV Studio for WebUI emulator:

Application screen with more than one event

Application screen with an event before and after the current event

Application screen with the last event

Application screen when the user changes language preference on the phone. The language preference can be modified by going to the Main Menu > Settings > Phone Settings > Language and Input > Language.

Summary

The article has demonstrated developing a simple WebUI application. The aspects covered include:

- Accessing information on the mobile phone using APIs exposed by the WebUI application environment. The API's covered are:
  - Calendar
  - Date
  - I18n
- Accessing an external web service
- Configuring Soft Key Menus
- Internationalization

The example gives shows about how easy it is to use WebUI for creating a semantically meaningful application which could appeal to a wide audience.

Glossary

API – Application Programming Interface

AJAX – Asynchronous JavaScript and XML

CSS – Cascading Style Sheets

Design Pattern - Solution or computing framework for a known problem

DOM – Document Object Model

UI – User Interface

W3C – World Wide Web Consortium

WDK – Web Development Kit

Web Service - Web services share business logic, data and processes through a programmatic interface across a network. The web service is an applications interface, not a user interface. Developers can then add the Web service to a GUI (such as a Web page or an executable program) to offer specific functionality to users.

WebUI - WebUI is an application environment for creating widgets and rich mobile web applications (WebUI Apps)

XML – eXtensible Markup Language

Appendix A

The ‘index.html’ file at the end of phase1:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
	<head>
		<meta http-equiv="Content-Type" content="text/html;
		charset=iso-8859-1">
		<link href="style/style.css" rel="stylesheet"
		type="text/css">
		<script src="scripts/script.js"
		type="text/javascript"></script>
		<script src="scripts/scriptEngine.js"
		type="text/javascript"></script>
	</head>
	<body onload="onLoadAction();">
		<div id="dateDisplay" style="display:block">
		<h4 id="dateConst">DATE</h4>
		<p id="dateFormat"></p>
		</div>

		<div id="timeDisplay" style="display:block">
		<h4 id="timeConst">TIME</h4>
		<p id="timeFormat"></p>
		</div>

		<div id="eventDisplay" style="display:block">
		<h4 id="eventConst">EVENT</script></h4>
		<p id="eventFormat"></p>
		</div>
	</body>
</html>
```

The ‘script.js’ file at the end of phase1:

```
var menu;   //Holds the soft key menu being choosen to be displayed

/* On Initial Display, fetch the list of Events and Display the first
Event on Screen */
function onLoadAction()
{
	println("In onLoadAction");

	getEventsList();
	println("get events list api called");
	println("Got list size= "+ noOfEvents);

	if (noOfEvents == 0)
	{
```

```
            TestAddEvents();
            getEventsList();
            println("get events list api called after adding test
      events");
            println("Got list size= "+ noOfEvents);
      }

      currEventCtr = 0;

      if (noOfEvents > 0)
      {
            displayScreen();
      }
      else
      {
            displayEmptyScreen();
      }
}

/* Handler Function to handle soft key events */
function handleSoftKeys(action)
{
      if (action == 'getNext')
      {
            currEventCtr = currEventCtr+1;
            displayScreen();
      }
      else if (action == 'getPrevious')
      {
            currEventCtr = currEventCtr-1;
            displayScreen();
      }
      else if (action == 'goExit')
      {
            webui.terminate(0);
      }
}

/* Display the Screen Populating the Event Details */
function displayScreen()
{
      document.getElementById("dateFormat").innerHTML =
      " "+getEventStartDate()+"/"
      +getEventStartMonth()+"/"
      +getEventStartYear();

      document.getElementById("timeFormat").innerHTML =
      " "+getEventStartHour()+":"
      +getEventStartMinute()+"  -  "
      +getEventEndHour()+":"
      +getEventEndMinute();
      getEventSummary();
      setTimeout('displayEvent()', 2000);
```

```
	println("nofevents = "+noOfEvents);
	println("currEventCtr = "+currEventCtr);

	if ((noOfEvents-1) > currEventCtr)
	{
		if (currEventCtr == 0)
		{
			println("displaying planner menu");
			// Set the Menu with Options to Display More Events
			menu = webui.softKeys.loadMenu("Planner");
		}
		else
		{
			println("displaying Planner2 menu");
			// Set the Menu with Options to Display More Events
			menu = webui.softKeys.loadMenu("Planner2");
		}
	}
	else
	{
		if (noOfEvents == 1)
		{
			println("displaying Planner4 menu");
			// Set the Simple Exit Menu
			menu = webui.softKeys.loadMenu("Planner4");
		}
		else
		{
			println("displaying Planner3 menu");
			// Set the Simple Exit Menu
			menu = webui.softKeys.loadMenu("Planner3");
		}
	}
	setMenus();
}

function displayEvent(translatedSummary)
{
	println("Translated Summary In disp Event="
	+translatedSummary);

	document.getElementById("eventFormat").innerHTML =
	translatedSummary;
}

/* Sets the chosen Menu for Display Purposes */
function setMenus()
{
	webui.softKeys.setActiveMenu(menu);
	listener = webui.softKeys.createListener();
	listener.handle = handleSoftKeys;
	menu.addListener(listener);
}
```

```
/* Displays a screen mentioning no events exist in the calendar */
function displayEmptyScreen()
{
	document.getElementById("dateFormat").innerHTML = “No Events”;
	document.getElementById("timeFormat").innerHTML = "No Events";
	document.getElementById("eventFormat").innerHTML = "No Events";

	// Set the Simple Exit Menu
	menu = webui.softKeys.loadMenu("Planner4");
	setMenus();
}
```

The file ‘scriptEngine.js’ at the end of phase1:

```
var eventsList;                // List of Events Holder
var noOfEvents = 0;            // Counter to know the Number of Events
var currEventCtr = 0;          // Counter to track the current Event


/* Fetch the list of Events in the next one week originating from the
current Time */
function getEventsList()
{
	var currDate, newDate;
	currDate = webui.system.util.time.getCurrentTime();
	newDate = webui.system.util.time.getCurrentTime();
	newDate.setDate(newDate.getDate()+7);
	eventsList = webui.calendar.getEvents(currDate, newDate);
	noOfEvents = eventsList.size;
}

/* Particular Event Details Fetching apis */

/* Fetch the Events Day of the Month */
function getEventStartDate()
{
	return eventsList[currEventCtr].getStart().getDate();
}

/* Fetch the Events Month of Occurence*/
function getEventStartMonth()
{
	// Month Values are 0-11 hence the addition of 1
	return ((eventsList[currEventCtr].getStart().getMonth()) + 1);
}

/* Fetch the Events Year of Occurence*/
function getEventStartYear()
{
	return eventsList[currEventCtr].getStart().getFullYear();
}
```

```
/* Fetch the Events Hour of Occurence */
function getEventStartHour()
{
	return eventsList[currEventCtr].getStart().getHours();
}

/* Fetch the Events Minute of Occurence */
function getEventStartMinute()
{
	return eventsList[currEventCtr].getStart().getMinutes();
}

/* Fetch the Events Hour of Completion */
function getEventEndHour()
{
	return eventsList[currEventCtr].getEnd().getHours();
}

/* Fetch the Events Minute of Completion */
function getEventEndMinute()
{
	return eventsList[currEventCtr].getEnd().getMinutes();
}

/* 1.Fetch the Events Summary Description.
   2.Perform the language conversion to the current Language Selected
on the Device.
*/

function getEventSummary()
{
	var summary = eventsList[currEventCtr].summary;
	println("summary = "+summary);
	displayEvent(summary);
}

/* Test Function to insert events in the emulator */
function TestAddEvents()
{
	var testEvt = webui.calendar.createEvent();
	var stdt = webui.system.util.time.getCurrentTime();
	stdt.setDate(stdt.getDate()+2);
	testEvt.setStart(stdt);
	stdt.setHours(stdt.getHours()+1);
	testEvt.setEnd(stdt);
	testEvt.summary = "test event one";

	var testEvt2 = webui.calendar.createEvent();
	stdt.setDate(stdt.getDate()+1);
	testEvt2.setStart(stdt);
	stdt.setHours(stdt.getHours()+2);
	testEvt2.setEnd(stdt);
	testEvt2.summary = "test event two";
```

```
        var testEvt3 = webui.calendar.createEvent();
        stdt.setDate(stdt.getDate()+2);
        testEvt3.setStart(stdt);
        stdt.setHours(stdt.getHours()+3);
        testEvt3.setEnd(stdt);
        testEvt3.summary = "test event three";

        webui.calendar.addEvent(testEvt, false);
        webui.calendar.addEvent(testEvt2, false);
        webui.calendar.addEvent(testEvt3, false);
}
```

The file ‘style.css’ at the end of phase1:

```
body
{
        background-color: #0f0faf;
}
#dateConst
{
        text-align:left;
}
#timeConst
{
        text-align:left;
}
#eventConst
{
        text-align:left;
}
#dateFormat
{
        text-align:center;
}
#timeFormat
{
        text-align:center;
}
#eventFormat
{
        text-align:center;
}
```

For Complete Source Code, Double click below:

DemoPlanner.zip