

### **DEVELOPING AN INTERNATIONALIZED WEBUI APPLICATION** V 1.0

**TECHNICAL ARTICLE** 



ECCN 5E991.NR: In accordance with United States Export Administration Regulations (EAR), and specifically the Commerce Control List (CCL), this item has been classified 5E991.NR. Export or re-export of this commodity and compliance with the U.S. Export Administration Regulations is ultimately the responsibility of the exporter. For more detailed information related to export or re-export of this item, please consult the EAR at http://www.access.gpo.gov/bis/ear/ear\_data.html.

Copyright © 2008, Motorola, Inc.

This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications, can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application.

Motorola makes no warranty in regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by applicable law. Otherwise, they are specifically excluded.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, the buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacturing of the product or service.

Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc.

MOTOROLA and the Stylized M Logo are registered in the US Patent & Trademark Office. Java and all other Javabased marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the US and other countries. All other product and service names are the property of their respective owners.

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		
	al application development	
	View Component	
	Model Component	
	Controller Component	8
	Soft Keys	
Phase 2: Inte	ernationalization	13
Phase 3: Util	lizing an external web service	16
Summary		21
Glossary		
Appendix A		24



## 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:

DATE

01/01/2009

TIME 17:00 - 17:30

EVENT

Fill in Petrol on way back home

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>
</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>
</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>
</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
{
<pre>text-align:left;</pre>
}
#timeConst
{
<pre>text-align:left;</pre>
}
#eventConst
{
<pre>text-align:left;</pre>
}

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
{
<pre>text-align:center;</pre>
}
#timeFormat
{
<pre>text-align:center;</pre>
}
#eventFormat
{
<pre>text-align:center;</pre>
}

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:



### 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"></screencontext>
<item></item>
<name>Next</name>
<action>getNext</action>
<item></item>
<name>Exit</name>
<action>goExit</action>

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:



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



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

💌 Soft Key Menu Design	1		
Soft Key Menu Design Menu Tree Define the elements of the Soft Key Me following section.		Soft Key Menu Details         Set the properties of the selected eleme         Required fields are denoted by "**",         Name:       Next         Action:       getNext	ent.
• • •			

Figure: Definition of Soft Key



fine the elements of the Soft Key Menu tree in the lowing section. Set the properties of the selected element. Required fields are denoted by "*". Id*: Planner Id*: Planner Id*: Planner	lowing section. Required fields are denoted by "*".	nu Tree		Soft Ke	ey Menu Details
New     ≥ Item       Remove     Remove       Run As     People       Debug As     People       Profile As     Validate       Team     Compare With	New     ≥ Item       Remove     Remove       Bun As     >       Debug As     >       Profile As     >       Validate     Tgam       Compare With     >       Replace With     >			Set the Require	properties of the selected element. ed fields are denoted by "*".
		3 <b>9</b> 1	New     Item       Remove     Remove       Bun As     Pebug As       Debug As     Profile As       Validate     Tgam       Tgam     Compgre With	Id*:	Planner

#### 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:

<pre>function handleSoftKeys(action)</pre>
{
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:

**Technical Article** 



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'



elect which	language translation	file you want to gen	erate:			
anguage				 ▼ Langu	age ID fr-ca	
ariation	Canadian					

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.

Activities solids from c		ttext Portable Object (.po)				1
Files:		Strings to externalize:				
🖂 🗹 🔛 Planner		Value	File	Line	Externalized	1
- 🗹 🙆 SkMenu.:	xml [9 occurence	Next	Sk/Menu.cml	9	No	Ш
- 🗹 🔝 index.htn	nl [3 occurences	Exit     Options     Next     Previous	Sk/Menu.xml	13	No	
Script.is	[38 occurences]	Options	Sk/Menu.xml	19	No	
ScriptEng	ine is [4 occurer	Mext	SkMenu.cml	22	No	
El 3 refress		Previous	SkMenu.xml	26	No	
		Exit	SKMenu.cml	31	No	
4	1 1	Previous	SKMenu.xml	37	No	
		Exit	SKMenu.xml	41	No	2
athe		Select All	Deselect All	Sel	oct Externalized	
1110000						
ource:					2	1

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.il8n.localize("DATE");

The webui.il8n 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",		
{				
callba	ck: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.reso	ources.get(	Curre	entLocale	e().langua	age;

• Perform the Google translate api call as follows:



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.

**Technical Article** 





When selected, the following screen is presented:

Run		
reate, manage, and run conf	gurations	
Image: system filter text       Image: system filter text	Name: DemoPlanner Man Project Type WebUI Application	T
JU Junit MOTODEV Application SemoPlanner SD SD SD WebUI Configuration (: Run on Server	Project Project Project Application / Package Planner	×
	Device MOTOMAGX Emulator	Configure
	Emulator Deployer Signing Studio_for_webui_emulator	Manage Certificates
Iter matched 10 of 12 items		Apply Reyert
0		Run Close

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);
```

**Technical Article** 



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
  - ≻ l18n
- 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"</pre>
           type="text/css">
           <script src="scripts/script.js"</pre>
           type="text/javascript"></script>
           <script src="scripts/scriptEngine.js"</pre>
           type="text/javascript"></script>
     </head>
     <body onload="onLoadAction();">
           <div id="dateDisplay" style="display:block">
           <h4 id="dateConst">DATE</h4>
           </div>
           <div id="timeDisplay" style="display:block">
           <h4 id="timeConst">TIME</h4>
           </div>
           <div id="eventDisplay" style="display:block">
           <h4 id="eventConst">EVENT</script></h4>
           </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)
    {
}
```

**Technical Article** 



```
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);
```







```
/* 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:

