How to build a Web Runtime layout with Header and Buttons Bar

Description

This article explains how to build from scratch a layout for Web Runtime widgets with these UI features:

- a header bar, that can contain the widget name and other optional elements
- a central body, containing the dynamic content of the widget
- a buttons' toolbar

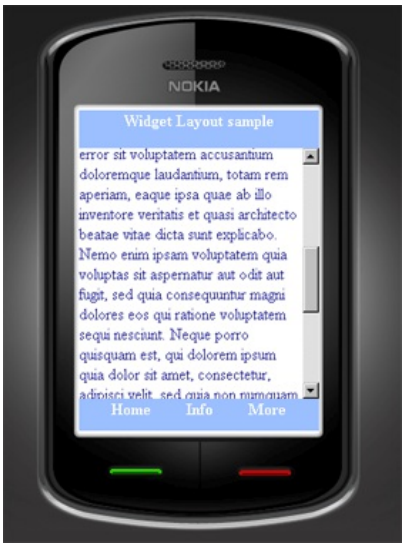
Also, the built layout will support the following functionalities:

- unified method to add buttons or softkeys, depending on the widget needs (e.g.: based on device's display resolution)
- support for display rotations
- repositioning of buttons' toolbar depending on display orientation
- support for easy switching of the widget main view
- automatic support for scrolling

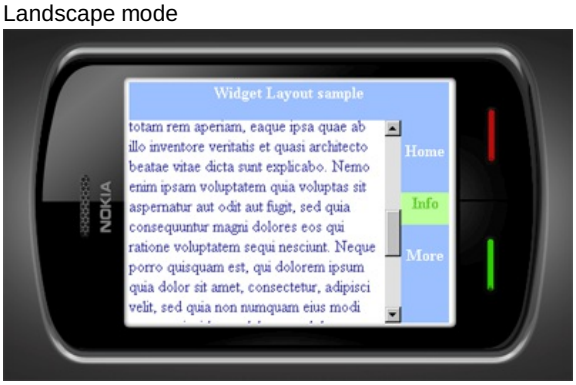
Sample layouts

The following pictures shows how the layout appears in portrait and landscape mode:

Portrait mode



Landscape mode



Implementation

The layout HTML structure

The layout is structured into these main DOM elements:

- a DIV element for the header
- a DIV element for the widget content
- 2 DIV elements for the buttons' toolbar, that will be used depending on the display orientation

The basic layout structure is the following:

(C) Copyright Nokia 2014. All rights reserved.

```
<html>
 <body>

  <div id="header">

  </div>

  <div id="right_toolbar">

  </div>

  <div id="screen_container">

  </div>

  <div id="bottom_toolbar">
   <div id="toolbar_buttons">
   </div>
  </div>


 </body>
</html>
```

The DIV element with toolbar_buttons id is a placeholder element for the toolbars' buttons.

The basic CSS code

In order to properly layout the various interface elements, very few basic CSS rules are needed:

```
#right_toolbar
{
 float:right;
}
#screen_container
{
 overflow-y:auto;
}
```

Things to note in the above CSS code:

- the `right_toolbar` is set to float: right, in order to have the toolbar, on landscape displays, on the right side of the widget. In order to have it on the left side, it is enough to change the float property to left.

An alternative approach to have the buttons' toolbar differently placed depending on the display orientation, based on absolute positioning, is visible in the STEW widget, and is explained in details here: STEW: supporting screen rotation

- the `overflow-y: auto` property for the `screen_container` allow to support vertical scrolling for views that are higher than the available space.

The JavaScript code

All the widget layout is managed through a layout JavaScript object, that will be defined and implemented in this section. This object takes care of layout appearance, events management, and view switching.

The layout properties

First, some properties are set in order to control the layout behavior and appearance.

```
var layout =
{
 /* true if the layout must use softkeys
  * false if the layout must use toolbar buttons
  */
```


(C) Copyright Nokia 2014. All rights reserved.

```
 useSoftkeys: false,

 /* CSS class applied to the buttons' toolbar */
 toolbarButtonCssClass: 'toolbar_button',

 /* value of the last detected display width */
 lastDetectedWidth: 0,

 /* value of the last detected display height */
 lastDetectedHeight: 0,

 /* holds a reference to the currently displayed view */
 currentScreen: null
}
```

Buttons/SoftKeys generation

Depending on the `useSoftkeys` property value, the layout will generate and handle softkey menu items or toolbar buttons, that will be inserted in the `toolbar_buttons` placeholder DIV element. In order to do this, the following `addMainOption()` method is defined:

```
var layout =
{
 [...]

 addMainOption: function(text, id, handler)
 {
  if(this.useSoftkeys)
  {
   var item = new MenuItem(text, id);

   item.onSelect = handler;

   menu.append(item);
  }
  else
  {
   var button = document.createElement('div');

   button.className = this.toolbarButtonCssClass;

   button.appendChild(document.createTextNode(text));

   button.onclick = handler;

   document.getElementById('toolbar_buttons').appendChild(button);
  }
 },
}
```

The three arguments passed to the `addMainOption()` method are:

- the menu item/button label,
- the menu item id (id is not used for toolbar buttons)
- the handler function called on the select/click events

Laying out the layout elements

First, a helper methods is defined in order to detect display orientation:

```
var layout =
{
```


(C) Copyright Nokia 2014. All rights reserved.

```
 [...]

 isLandscape: function()
 {
  return window.innerWidth > window.innerHeight;
 }
```

The layout object has to manage the widget's appearance, and specifically has to perform these operations:

- resize the various layout elements depending on the display size and orientation
- place the toolbar buttons (if not using softkeys) according to the display orientation

All these operations are managed by the following setupInterface() method:

```
var layout =
{
 [...]

 setupInterface: function()
 {
  if(!this.useSoftkeys)
   menu.hideSoftkeys();

  this.lastDetectedWidth = window.innerWidth;
  this.lastDetectedHeight = window.innerHeight;

  if(this.useSoftkeys)
  {
   document.getElementById('screen_container').style.height =
    (this.lastDetectedHeight - document.getElementById('header').offsetHeight) + 'px';
  }
  else if(this.isLandscape())
  {
   document.getElementById('bottom_toolbar').style.display = 'none';
   document.getElementById('right_toolbar').style.display = '';

document.getElementById('right_toolbar').appendChild(document.getElementById('toolbar_buttons'));

   document.getElementById('right_toolbar').style.height =
    (this.lastDetectedHeight - document.getElementById('header').offsetHeight) + 'px';

   document.getElementById('screen_container').style.height =
    (this.lastDetectedHeight - document.getElementById('header').offsetHeight) + 'px';
  }
  else
  {
   document.getElementById('bottom_toolbar').style.display = '';
   document.getElementById('right_toolbar').style.display = 'none';

document.getElementById('bottom_toolbar').appendChild(document.getElementById('toolbar_buttons'));

   document.getElementById('screen_container').style.height =
    (this.lastDetectedHeight - document.getElementById('header').offsetHeight -
document.getElementById('toolbar_buttons').offsetHeight) + 'px';
  }
 }
}
```

Managing display rotations

When the display changes orientation, the interface needs to be relayed out, in order to resize and move elements appropriately. Since all these things are performed by the `setupInterface()` method, the following `onResize()` method will uniquely call this

 (C) Copyright Nokia 2014. All rights reserved.

method:

```
var layout =
{
 [...]

 onResize: function()
 {
  this.setupInterface();
 },
}
```

Setting the widget main view

In order to easily change the main view of the widget, the `gotoScreen()` is defined, that will hide the currently visible view, and show the new one.

```
var layout =
{
 [...]

 gotoScreen: function(screenId, clearHistory)
 {
  var newScreen = document.getElementById(screenId);

  if(newScreen)
  {
   if(this.currentScreen != null)
   {
    this.currentScreen.style.display = 'none';
   }
   this.currentScreen = newScreen;

   this.currentScreen.style.display = '';
  }
 }
}
```

Initializing layout and events

Now, all is ready to be initialized and used. So, it is possible to define an `init()` method that will take care of:

- calling methods for layout initialization
- setting up event handlers for widget resize events

```
var layout =
{
 [...]

 init: function(startScreen)
 {
  this.setupInterface();

  var self = this;

  window.addEventListener(
   'resize',
   function()
   {
    self.onResize();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    },
    false
   );

   this.gotoScreen(startScreen);
  }
 }
```

Polling for display rotations

Some devices, as explained here, do not support the 'onResize' event. So, the approach defined in the linked Nokia Developer Library page is implemented, in order to correctly manage display resize events also on these devices. The following `pollResize()` method is defined:

```
var layout =
{
 [...]

 pollResize: function()
 {
  if(window.innerWidth != this.lastDetectedWidth || window.innerHeight !=
this.lastDetectedHeight)
  {
   this.onResize();
  }
 }
}
```

And is scheduled to be periodically called by the init() method, that is modified as follows:

```
var layout =
{
 [...]

 init: function(startScreen)
 {
  [...]

  setInterval(
   function()
   {
    self.pollResize();
   }
   ,
   1000
  );
 }
}
```

Layout object: how to use it

Usage of the layout object is quite straightforward, and requires these steps:

- import the layout JavaScript and CSS files (available here: Media:WRTLayoutHeaderAndButtons JSandCSS.zip

) in your widget's HTML code:

```
<html>
 <head>
  <script language="javascript" type="text/javascript" src="layout.js"></script>
```

 (C) Copyright Nokia 2014. All rights reserved.

```
  <link rel="stylesheet" href="layout.css" type="text/css">
 </head>

 [...]
</html>
```

- define the basic HTML structure, as defined in the above section, and complete them with your widget specific code. An example of widget HTML code with 3 main views is the following:

```
<html>
 [...]

 <body onLoad="javascript:init();">



  <div id="header">
   Widget Layout sample
  </div>

  <div id="right_toolbar">
  </div>

  <div id="screen_container">

   <div class="screen" id="screen_home" style="display: none;">
    <strong>Widget Layout with Header and Buttons' Toolbar</strong>
   </div>

   <div class="screen" id="screen_info" style="display: none;">
    <h2>Widget Info</h2>

    Some infos about the application
   </div>

   <div class="screen" id="screen_more" style="display: none;">
    Some more info...
   </div>
  </div>

  <div id="bottom_toolbar">
   <div id="toolbar_buttons">

   </div>
  </div>


 </body>
</html>
```

- then, the layout can be initialized, by adding the desired softkeys/buttons and by calling its initialization method.

```
function init()
{
 layout.useSoftkeys = false;

 layout.addMainOption('Home', 1001, function(){layout.gotoScreen("screen_home");})
 layout.addMainOption('Info', 1002, function(){layout.gotoScreen("screen_info");})
 layout.addMainOption('More', 1003, function(){layout.gotoScreen("screen_more");})
```


(C) Copyright Nokia 2014. All rights reserved.

```
  layout.init('screen_home');
 }
```

So this example layout does not use softkeys: 3 buttons are so created, and each of them brings to a view defined in the above HTML code. Screenshots of the sample widget are visible at the beginning of this article.

Downloads

The following related files are available for download:

- A sample widget using the layout implemented in this article: Media:WRTLayoutHeaderAndButtons.zip
- The layout JavaScript and CSS files: Media:WRTLayoutHeaderAndButtons JSandCSS.zip

(C) Copyright Nokia 2014. All rights reserved.