Creating a Slider component for Mobile Web Templates

This article explains how to create and use a **Slider UI component** for Mobile Web Templates for High-End devices.

Introduction

Mobile Web Templates are a set of HTML resources that allow to **easily build Web pages especially suited for mobile devices**.

Mobile Web Templates for High-End devices contain some **JavaScript components** that allow to create and use interactive elements within a mobile web page. The available components are:

- **Slideshow**: allows to **visualize a set of images**, by showing a single image and adding **buttons to navigate** to the previous and next photos
- **Toggle**: a more usable and **good-looking alternative to standard checkboxes**, allowing users to enable or disable a particular option
- **Accordion**: allows to build **collapsible lists**, especially useful on mobile devices' displays, where available space is usually limited

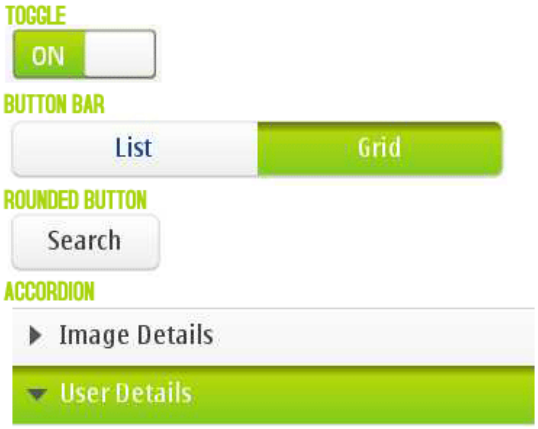
The Slider component

This article will focus on building a new component for Mobile Web Templates: a Slider component. The Slider is a **control that enables the user to select a value in a finite range**. A full description of the **Slider design pattern** is available here: Mobile Design Pattern: Slider Control

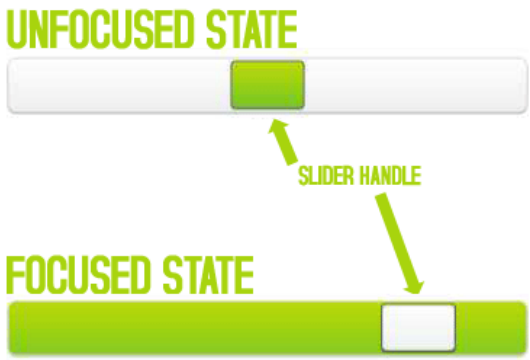
The next sections show the involved **designing and programming steps**, explaining the more relevant CSS, HTML and JavaScript code portions. **Compatibility aspects with both S60 3rd edition and 5th edition devices** will be covered, in order to build a component compatible with both touch and non-touch devices. Also, **graceful degradation will be implemented**, so that browsers without JavaScript support will be able to render a "scaled-down", but usable, version of the component itself.

Designing the component

Let's start by **designing the component, following the default style of Mobile Web Templates** for High-End devices. The following picture shows some of the **inbuilt components**:



So, using the same style and colors, it is possible to **design the Slider component** like the following one:



The Slider will basically be a **horizontal bar, with a handle representing the current value**.

The scaled down version

On browsers without JavaScript support, it is necessary to offer to the user a **plain-HTML version of the component, that does not rely on**

(C) Copyright Nokia 2013. All rights reserved.

JavaScript for its functionality. The most immediate and effective element that can be used as a Slider "degraded" version is represented by a **text field, that allows the user to enter the value he wants** by directly typing the value. The plain-HTML version is visible below.

Select width
30

Coding the component

The plain-HTML version

First thing to do, is to **start from the plain-HTML version of the component**, that will be directly inserted in the web page HTML code. The following snippet shows a text field with an associated label that suit this requirements:

```
<label for="width_slider">Select width</label>
<input type="text" name="width_slider" id="width_slider" />
```

This INPUT field is the starting point to build the Slider, as it will be dynamically replaced by a fully interactive JavaScript component.

The constructor

The following properties must be defined, in order to properly initialize a Slider component:

- the **DOM element** to be replaced
- the **minimum input value**
- the **maximum input value**
- the **starting input value**
- the **increment** unit step (1 by default)

Also, defining a **callback can be useful to be notified when the slider value changes**, in response to actions of the user.

It is then possible to **define a constructor** with as the following:

```
function Slider(_id, _defaultValue, _minValue, _maxValue, _increment, _callback)
{
 this.id = _id;
 this.sliderInput = document.getElementById(_id);

 this.value = (_defaultValue ? _defaultValue : 0);
 this.minValue = (_minValue ? _minValue : 0);
 this.maxValue = (_maxValue ? _maxValue : 100);
 this.increment = (_increment ? _increment : 1);

 this.callback = _callback;

 (this.sliderInput) ? this.init() : false;
}
```

Given the HTML snippet defined above, it is possible to create a Slider component with the following code:

```
var mySlider = new Slider('width_slider', 30, 20, 40, 2, sliderHandler);
```

DOM structure

The DOM structure chosen for the Slider is the following one:

```
<ul class="slider">

 <!-- the slider right bound -->
 <li class="slider-item">

  <!-- the slider left bound -->
  <a href="#">

   <!-- the slider handler -->
   <span style="background-position: 50% center;"></span>
  </a>
 </li>

 <!-- a helper element used to display the current slider value -->
 <li class="slider-value">
  <p>
   <!-- here the current value goes -->
  </p>
 </li>
</ul>
```

Some notes about the above structure:

- **left and right bounds** of the slider are added with 2 nested elements: this allows to have a nicely resizable component, that scales well on all displays resolutions. This is also obtained with the following CSS rules:

```
ul.slider li.slider-item {
 background: url(../images/sprite-button-rounded-right.png) no-repeat right top;
}
ul.slider li.slider-item:hover {
 background: url(../images/sprite-button-rounded-right.png) no-repeat right bottom;
}
ul.slider li.slider-item a {
```


(C) Copyright Nokia 2013. All rights reserved.

```
 background: url(../images/sprite-button-rounded-left.png) no-repeat left top;
}
ul.slider li.slider-item a:hover {
 background: url(../images/sprite-button-rounded-left.png) no-repeat left bottom;
}
```

- **the "a" element** is used to allow 3rd edition (non-touch) devices to properly focus (and then use) the component

Component initialization

If the constructor can properly find the base input text field, then the **initialization function, init()**, is called.

The **initialization process** has to build the component DOM structure seen above, setting the necessary **JavaScript event handlers** to add the proper behavior to the various elements.

```
Slider.prototype.init = function()
{
 //hide the text input
 this.sliderInput.style.display = 'none';

 //create the slider <ul> element
 this.slider = document.createElement('ul');
 this.slider.setAttribute('class', 'slider');

 //create the slider <li> element
 var li = document.createElement('li');
 li.setAttribute('class', 'slider-item');

 //create the <a> element
 var aElement = document.createElement('a');
 aElement.href = "#";

 //create the <span> element (the slider handle)
 this.sliderHandle = document.createElement('span');

 //create the slider value <li> element
 var valueLi = document.createElement('li');
 valueLi.setAttribute('class', 'slider-value');

 //create the value element
 this.sliderValueElement = document.createElement('p');
 this.sliderValueElement.innerHTML = this.value;

 //appends the elements
 aElement.appendChild(this.sliderHandle);
 li.appendChild(aElement);
 this.slider.appendChild(li);

 valueLi.appendChild(this.sliderValueElement);
 this.slider.appendChild(valueLi);

 //replace the input with the slider element
 this.sliderInput.parentNode.insertBefore(this.slider, this.sliderInput);

 var self = this;

 // event for non-touch devices
 aElement.onkeydown = function(e)
 {
  self.handleKeyDown(e);
 }

 // event for touch devices
 this.sliderHandle.onclick = function(e)
 {
  self.handleClicked(e);
 }

 //set initial slider's value
 this.setValue(this.value);
}
```

The update behavior

First, it is necessary to define a function that is called every time the **value of the Slider needs to be changed**. This can be done as follows:

```
Slider.prototype.setValue = function(_value)
{
 this.sliderValueElement.innerHTML = _value;

 this.value = _value;

 this.update();

 if(this.callback)
 {
  this.callback(this);
 }
}
```

Then, it is necessary to create a **function that updates the Slider's appearance (by moving the Slider handle)** to represent the currently selected value.

```
Slider.prototype.update = function()
{
 var positionX = 100 * (this.value - this.minValue) / (this.maxValue - this.minValue);

 this.sliderHandle.style.backgroundPosition = positionX + '% center';

 this.sliderInput.value = this.value;
}
```


(C) Copyright Nokia 2013. All rights reserved.

Handling key events

Non-touch devices, where tabbed navigation is enabled, allow for **key-based interactions**. Then, the component has to be able to **intercept key events**, and to properly respond to them.

The desired behavior is the following one:

- when the **right key** is pressed, the slider value has to be **incremented**
- when the **left key** is pressed, the slider value has to be **decremented**

The following code implements this behavior:

```
Slider.prototype.handleKeyDown = function(e)
{
 var key = e.keyCode;

 var newValue = this.value;

 if(key == 37 && this.value > this.minValue)
  newValue = Math.max(this.minValue, this.value - this.increment);
 else if(key == 39 && this.value < this.maxValue)
  newValue = Math.min(this.maxValue, this.value + this.increment);

 if(newValue != this.value)
  this.setValue(newValue);
}
```

Handling touch events

Touch devices, and non-touch devices with **cursor-based navigation** enabled, allow for **touch/click based interactions**. The component has then to be able to intercept and respond to click events, by **using the click X coordinate** to set the appropriate new value for the Slider.

The following code implements this functionality:

```
Slider.prototype.handleClicked = function(e)
{
 var mouseX = e.clientX;

 var relativeX = mouseX - this.sliderHandle.offsetLeft;

 var newValue = this.minValue + Math.round((this.maxValue - this.minValue) * relativeX / (this.sliderHandle.offsetWidth * this.incre

 this.setValue(newValue);
}
```

Using the Slider

Usage of the Slider component is quite straightforward:

- Define the **plain-HTML version** of the component:

```
<input type="text" name="width_slider" id="width_slider" />
```

- Call the **Slider constructor**:

```
var mySlider = new Slider('width_slider', 30, 10, 100, 5, sliderHandler);
```

- To retrieve the Slider's value, simply use the **Slider "value" property**, or the **INPUT field value** (to guarantee graceful degradation):

```
// using the "value" property
var sliderValue = mySlider.value;

// using the INPUT field
var sliderValue = document.getElementById('width_slider').value;
```

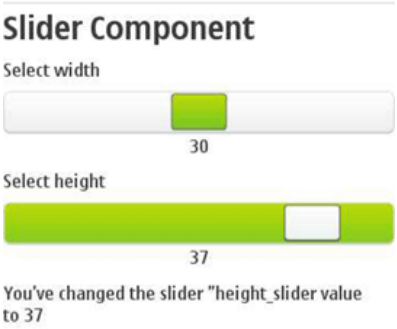
- Optionally, define the **slider callback**:

```
function sliderHandler(slider)
{
 alert('You\'ve changed the slider "' + slider.id + ' value to ' + slider.value);
}
```

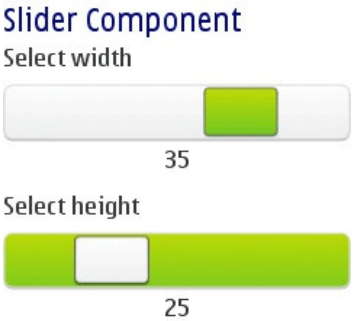
Screenshots

The following pictures show the component in action on **S60 5th edition devices**:

(C) Copyright Nokia 2013. All rights reserved.



and on **S60 3rd edition FP2 devices**:



Downloads

It is possible to download the Slider component and some usage examples from the following links:

- The **Slider component** (JavaScript, CSS, images): http://www.jappit.com/uploads/wrt/MWTSliderComponent.zip
- A **Web Runtime widget** that uses the Slider component: http://www.jappit.com/uploads/wrt/MWTSlider.wgz

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.