



This article explains **how to create custom scrollbars in a Web Runtime widget**. The technique presented in this article is **tested and working on S60 5th edition devices**.

Contents

- [1 Description](#)
- [2 How it works](#)
- [3 The code](#)
 - ◆ [3.1 HTML structure](#)
 - ◆ [3.2 The CSS code](#)
 - ◆ [3.3 Resizing the slider](#)
 - ◆ [3.4 Moving the slider](#)
- [4 Download](#)

Description

When presenting content that overflows the visible area, **scrollbars are usually shown to notify the user about the availability of extra content**, that overflows the visible one. A detailed description of the **Scrollbar Mobile Design Pattern** is available here: [Mobile Design Pattern: Scrollbar](#).

How it works

The approach used in this article is similar to the one described in the [Hiding default scrollbars in Web Runtime widgets](#) article, where **ad-hoc HTML elements are used to cover the default widget's default scrollbars**. Also, a **scroll event handler** is used in order to **correctly manage the scrollbar's slider position**.

The code

The following code shows **how to replace the default vertical scrollbar with a custom one**. Similar technique can be used to override the horizontal scrollbar.

HTML structure

The basic HTML structure is composed of **4 elements**:

- a **container element**
- the **element that holds the content to be scrolled**
- the **element that will simulate the scrollbar**
- and, inside the scrollbar element, a the **element representing the slider**

The sample HTML structure is shown in the following HTML snippet:

```
<html>
  [...]

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

  <h2>Custom scrollbar example</h2>

  <div id="container">

  <div id="scrollable_content">

  </div>

  <div id="scrollbar">
  <div id="slider"></div>
  </div>

  </div>

  </body>
</html>
```

The CSS code

In order to have the **scrollbar correctly positioned and styled**, as well as the slider element, it's necessary to add some **CSS rules**. Since the custom scrollbar has to replace the **default vertical scrollbar**, it has to be **positioned on the right side** of the content div, and has to take its full height. The following CSS rules allow to have a **custom 30-pixels wide scrollbar with blue background, and a rectangular gray slider**. **Further styling** can be performed in order to have **good looking scrollbars**, by using **custom images** placed within the scrollbar and slider elements.



```
#container {
position: relative;
overflow: hidden;
width: 200px;
}

#scrollable_content {
overflow: auto;
height: 200px;
padding-right: 24px;
}

#scrollbar {
background: blue;
height: 100%;
width: 30px;
position: absolute;
right: 0px;
top: 0px;
}
#slider {
background: gray;
position: absolute;
top: 0px;
left: 0px;
width: 100%;
}
```

Some notes about the above CSS code:

- In order to **allow scrolling**, the **"overflow"** property of the *scrollable_content* DIV element has to be set to **"auto"**.

- Since **default scrollbar width of WRT widgets is 4 pixels**, it's necessary to have to set an **explicit right-padding** to the scrollable element, in order to **avoid to cover the content with the scrollbar element**

Resizing the slider

Now that the scrollbar and its slider are correctly shown above the default ones, **the slider has to be resized and positioned according to the size of the scrollable content**: more is the content that overflows the visible area, smaller will be the slider height. In order to calculate the slider height, the **getSliderHeight()** function is defined:

```
function getSliderHeight()
{
var scrollable = document.getElementById('scrollable_content');

return scrollable.offsetHeight * scrollable.offsetHeight / scrollable.scrollHeight;
}
```

The **getSliderHeight()** function is used in the **init()** method to set the correct slider height:

```
function init()
{
var slider = document.getElementById('slider');

    slider.height = getSliderHeight() + 'px';
}
```

Moving the slider

When the user scrolls up and down the scrollable content, the **slider has to be moved to represent the actual position of the visible area** within the whole available content. The following **getSliderTop()** function returns the **top coordinate** of the slider, according to the actual visible area:

```
function getSliderTop()
{
var scrollable = document.getElementById('scrollable_content');

return scrollable.offsetHeight * scrollable.scrollTop / scrollable.scrollHeight;
}
```

The **slider position has to be updated each time the user scrolls** the content. So, the **onscroll** event can be used to detect such events, and it's possible to **modify the init() method**, defined above, as follows:

```
function init()
{
[...]
```

```
var scrollable = document.getElementById('scrollable_content');

    scrollable.onscroll = moveSlider;
}
```

How_to_create_custom_scrollbars_in_Web_Runtime_widgets

The **moveSlider()** method is the one that actually **updates the slider position**, by using the **getSliderTop()** method defined above:

```
function moveSlider()
{
var slider = document.getElementById('slider');

    style.top = getSliderTop() + 'px';
}
```

Download

The sample widget presented in this article is available for download here: [Media:CustomScrollbarWidget.zip](#)