Hiding default scrollbars in Symbian Web Runtime

This article explains how to hide scrollbars in a Web Runtime widget. This approach is tested working on S60 5th edition devices.

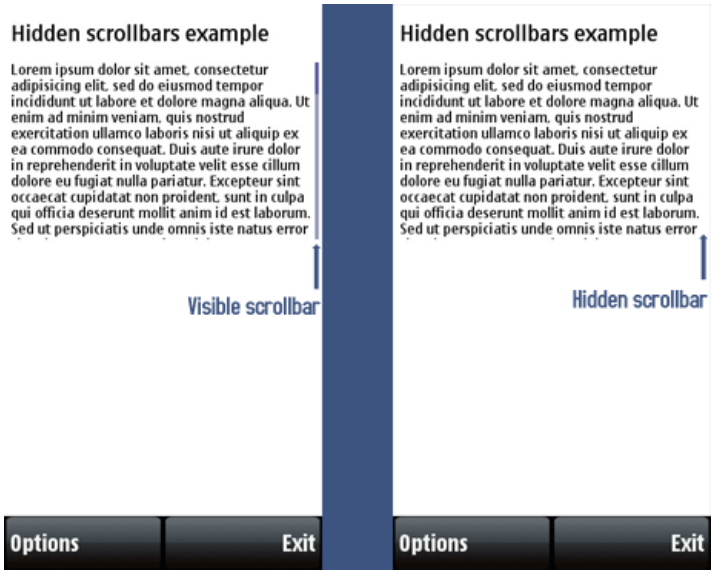
Description

Scrollbars are a common user interface element that allow to notify users about the presence of content that overflows the current visible area. A full description of the Scrollbar Mobile Design Pattern is available here: Mobile Design Pattern: Scrollbar Hiding scrollbars could be useful in order to allow further customization of a widget's layout. When hiding scrollbars, it is always good practice to give to users an alternative indicator of the overflowing content.

How it works

The approach described in this article works by "hiding" the scrollbars with a HTML element placed just above the scrollbars. Using appropriate colors and positioning, the cover element can perfectly hide the scrollbars, so giving the impression of absence of scrollbars.

The following code shows how to hide the vertical scrollbar, but the same technique can be used to hide the horizontal scrollbar as well.



The code

HTML structure

The basic HTML structure required to hide scrollbars is composed by 3 elements:

- a container element
- the actual content element
- the "cover" element

The basic structure is shown in the following HTML code snippet

```
<html>
 [...]

 <body>

  <div id="container">

   <div id="scrollable_content">
    Here goes the content
   </div>
```


(C) Copyright Nokia 2014. All rights reserved.

```
    <div id="scrollbar_cover" class="scrollbar_cover">
     <!-- This must be empty! -->
    </div>

   </div>

  </body>
 </html>
```

CSS styling

Now, the HTML elements need to be styled in order to have the scrollbar cover placed just over the scrollbar, by using absolute positioning. The three main elements are styled as follows.

- Styling the container element: Styling for the container element is pretty simple, and just requires to have the position: relative rule specified, in order to allow exact positioning of children nodes

```
#container {
 position: relative;
}
```

- **styling the content node**: in order to have scrollable content, the `overflow: auto` rule must be defined. Also, an explicit height is defined, in order to let the content overflow it.

```
#scrollable_content {
 overflow: auto;
 height: 200px;
}
```

- **styling the scrollbar cover**: the scrollbar cover must be exactly positioned over the default scrollbar. Standard S60 browser's scrollbars are 4 pixels wide, so we can assume and use this value for the CSS rules. The cover background color is set equal to the content's background color (white in this example).

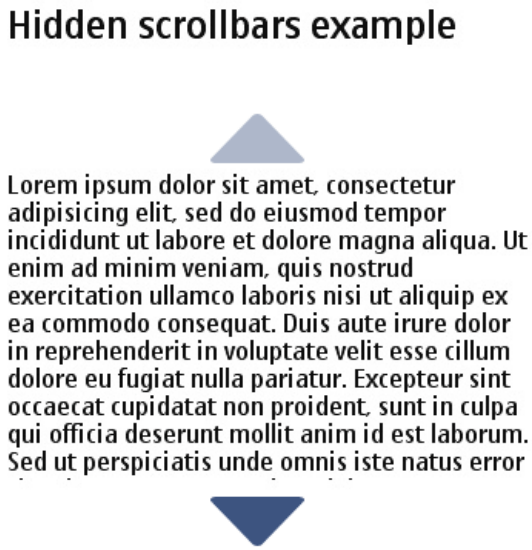
```
.scrollbar_cover {
 background: white;
 height: 100%;
 width: 4px;
 position: absolute;
 right: 0px;
 top: 0px;
}
```

Once these steps are done, the vertical scrollbar is actually hidden, and the user will no more see it.

Use custom scroll indicators

Now that the default scrollbar is hidden, it's highly recommended to use some sort of custom scroll indicator, so that the user knows that more content than the visible one is available. An example of custom scroll indicators is visible below:

(C) Copyright Nokia 2014. All rights reserved.



Further development

Dynamically retrieving scrollbar width

We've assumed the S60 browser's scrollbars to be 4 pixel wide, and the above code works well with this value. Anyway, it is also possible to retrieve the scrollbars' size by using JavaScript, and then to set the cover width with the value so obtained. In order to programmatically get the scrollbar width, the code presented on this page can be used:

```
function getScrollerWidth() {
    var scr = null;
    var inn = null;
    var wNoScroll = 0;
    var wScroll = 0;

    // Outer scrolling div
    scr = document.createElement('div');
    scr.style.position = 'absolute';
    scr.style.top = '-1000px';
    scr.style.left = '-1000px';
    scr.style.width = '100px';
    scr.style.height = '50px';
    // Start with no scrollbar
    scr.style.overflow = 'hidden';

    // Inner content div
    inn = document.createElement('div');
    inn.style.width = '100%';
    inn.style.height = '200px';
```


(C) Copyright Nokia 2014. All rights reserved.

```
    // Put the inner div in the scrolling div
    scr.appendChild(inn);
    // Append the scrolling div to the doc
    document.body.appendChild(scr);

    // Width of the inner div sans scrollbar
    wNoScroll = inn.offsetWidth;
    // Add the scrollbar
    scr.style.overflow = 'auto';
    // Width of the inner div width scrollbar
    wScroll = inn.offsetWidth;

    // Remove the scrolling div from the doc
    document.body.removeChild(
        document.body.lastChild);

    // Pixel width of the scroller
    return (wNoScroll - wScroll);
}
```

What the `getScrollerWidth()` does is to append and remove some DOM elements to find out which is the exact scrollbar width. Once obtained, this value can be used to set the cover width this way:

```
var scrollbarWidth = getScrollerWidth();

document.getElementById('scrollbar_cover').style.width = scrollbarWidth + 'px';
```

Recovering the scrollbar space

With the approach described above the scrollbar gets actually hidden, but its physical space is still taken, and so the scrollable content is a bit less wide than it could. In order to recover these pixels, it is possible to programmatically modify the width of the container element. Using the getScrollerWidth() function just defined:

```
var scrollbarWidth = getScrollerWidth();

document.getElementById('container').style.width =
 (document.getElementById('container').offsetWidth + scrollbarWidth) + 'px';
```

Downloads

A sample widget, showing the code implemented in this article, is available here: Media:HideScrollbarsWidget.zip

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.