Progress Bar JavaScript component for Web Runtime

This article explains how to create and use a Progress Bar component in Web Runtime widgets. A Progress Bar could be useful during long-running operations, to give a visual feedback to the user about the operation status. Two types of Progress Bars will be handled by this component: with definite and indefinite ranges.

The "Wait and Progress indicator design pattern" is detailed described by this Nokia Developer Wiki article: Mobile Design Pattern: Progress and Wait Indicator.

For more information about progress bars' usage, and guidelines about actions and visual feedback to the user, check out the *Actions and feedback* section of the **Design and User Experience Library**:

http://library.developer.nokia.com/index.jsp?topic=/Design_and_User_Experience_Library/GUID-A34281EC-A40A-403F-A3E1-74E7C9BD2784.html

ProgressBar component: how to use it

To use the Progress Bar component presented in this article, follow these steps:

- **Import the ProgressBar.js** JavaScript file in your Widget (source code available here: Media:Wrt ProgressBar component.zip):

```
<script language="javascript" type="text/javascript" src="ProgressBar.js"></script>
```

- Include in your project **2 images, of the same size**, for the **empty and full progress bar**, respectively. Sample images, used in this article, are the following:

- Define a **parent HTML node** for the component in your Widget HTML code:

```
<html>
[...]
<div id="loading_bar"></div>
[...]
</html>
```

- Finally, **create a ProgressBar instance**, passing these arguments to its constructor:
  - the empty bar image path
  - the full bar image path
  - the progress bar width

(C) Copyright Nokia 2014. All rights reserved.

- the progress bar height

And then **append the newly created instance to the HTML parent node** defined above:

```
progressBar = new ProgressBar('images/progress_empty.png', 'images/progress_full.png',
200, 28);

progressBar.appendTo(document.getElementById('loading_bar'));
```

ProgressBar component implementation

This part of the articles shows how the ProgressBar JavaScript component is actually implemented.

ProgressBar constuctor

The first part to implement is the **ProgressBar constructor**, whose arguments were already shown in the usage part.

```
function ProgressBar(baseImagePath, fullImagePath, barWidth, barHeight)
{
 /* DOM elements */
 this.fullImageElement = null;
 this.domElement = null;

 /* Progress Bar size */
 this.barWidth = barWidth;
 this.barHeight = barHeight;

 /* loading progress */
 this.progressValue = 0;

 this.init(baseImagePath, fullImagePath, barWidth, barHeight);
}
```

The costructor initializes the size-related properties of the ProgressBar, sets the **initial progress status to zero**, and then calls the **init() method** that will builds up the component DOM structure.

Building and using component's DOM structure

The whole **component DOM structure** will be built by the **init()** instance method, that is implemented as follows:

```
ProgressBar.prototype.init = function(baseImagePath, fullImagePath, barWidth, barHeight)
{
 var container = document.createElement('div');
 container.style.position = 'relative';
 this.domElement = container;

 var baseImage = document.createElement('img');
 baseImage.src = baseImagePath;
 baseImage.style.position = 'absolute';
 container.appendChild(baseImage);

 var fullImageContainer = document.createElement('div');
 fullImageContainer.style.position = 'absolute';
 fullImageContainer.style.overflow = 'hidden';
 fullImageContainer.style.height = barHeight + 'px';
 fullImageContainer.style.width = 0;
 container.appendChild(fullImageContainer);
```


(C) Copyright Nokia 2014. All rights reserved.

```
 this.fullImageElement = fullImageContainer;

 var fullImage = document.createElement('img');
 fullImage.src = fullImagePath;
 fullImage.style.position = 'absolute';
 fullImageContainer.appendChild(fullImage);
}
```

The component is structured as follows:

- an **outer DIV** element, that contains all the component
- the **empty bar image**, completely shown
- an **inner DIV, containing the full bar image**, that initially has width equal to zero. Having the 'overflow' CSS property set to 'hidden', the **full bar image will be initially hidden**, and will be partially or totally shown proportionally to the progress status.

To append the component main DOM element to a parent element defined in a widget, the following **appendTo()** method is defined:

```
ProgressBar.prototype.appendTo = function(parentElement)
{
 parentElement.appendChild(this.domElement);
}
```

Updating the progress status

The following **setLoadingProgress()** method manages the updates of the progress bar. It **accepts as argument the new progress value**, that can be any numeric value between 0 and 100, and then updates both the **progressValue** property, and the **UI status of the component**.

```
ProgressBar.prototype.setLoadingProgress = function(value)
{
 if(value < 0)
  value = 0;
 else if(value > 100)
  value = 100;

 this.progressValue = value;

 this.fullImageElement.style.width = (this.barWidth * value / 100) + 'px';
}
```

To update the component UI status, the **width of the fullImageElement** (the DOM element containing the full bar image) is **changed proportionally to the new progress value**.

Indefinite progress bars

The **setLoadingProgress()** is useful if you want to show the exact progress status to the user. But, there are situations where the **exact progress status is not known**, and the only thing available information is that the long running operation is still running. In these cases, it is **useful to show a progress bar that is always animated**, so restarting when it reaches the full status.

First, let's define **two ProgressBar properties** that will be used to handle the indefinite progress bars:

```
function ProgressBar(baseImagePath, fullImagePath, barWidth, barHeight)
{
 [...]

 /* progressbar type */
```


(C) Copyright Nokia 2014. All rights reserved.

```
  this.indefiniteProgress = false;
  this.indefiniteInterval = null;
 }
```

Then, the functionality is implemented through the following **setIndefinite()** method:

```
ProgressBar.prototype.setIndefinite = function(isIndefinite)
{
 if(this.indefiniteProgress != isIndefinite)
 {
  this.indefiniteProgress = isIndefinite;

  if(!this.indefiniteProgress)
  {
   clearInterval(this.indefiniteInterval);

   this.indefiniteInterval = null;
  }
  else
  {
   var self = this;

   this.indefiniteInterval = setInterval(
    function()
    {
     var nextValue = self.progressValue + 2;

     if(nextValue > 100)
      nextValue -= 100;

     self.setLoadingProgress(nextValue);
    },
    100
   );
  }
 }
}
```

If the **isIndefinite** argument is true, then the progress bar is treated as indefinite, and an always running animation is started. Otherwise, if it is false, the animation, if running, is stopped.

Downloads

The following files, used in this article, are available for download:

- A sample WRT widget using the ProgressBar component: Media:ProgressBarWidget.zip
- The ProgressBar component source code: Media:Wrt ProgressBar component.zip

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.