



This article explains how to implement the **Live Scrolling** Design Pattern in a Web Runtime widget.

The **Live Scrolling Design Pattern** is explained in detail in this Forum Nokia Wiki Article: [Mobile Design Pattern: Live Scrolling](#)

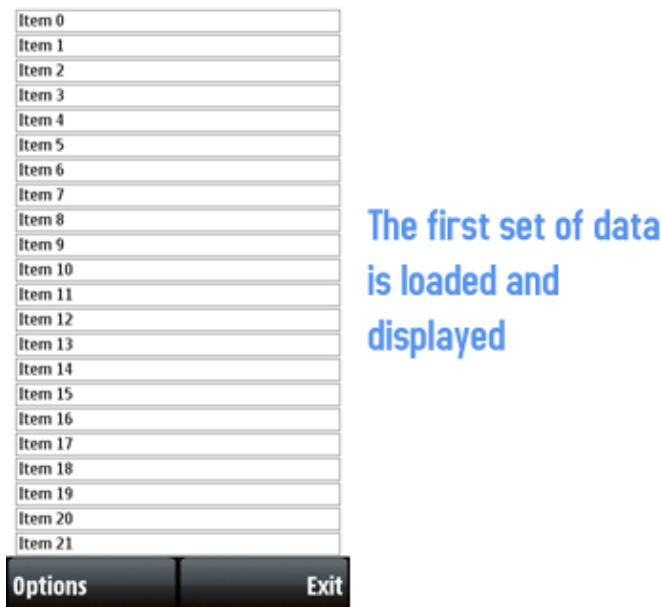
Contents

- [1 Description](#)
- [2 How to load remote data](#)
- [3 Loading and displaying chunks of data](#)
- [4 Notifying the user with a loading indicator](#)
- [5 Handling the scroll events](#)
- [6 Loading the first set of data](#)
- [7 Downloads](#)

Description

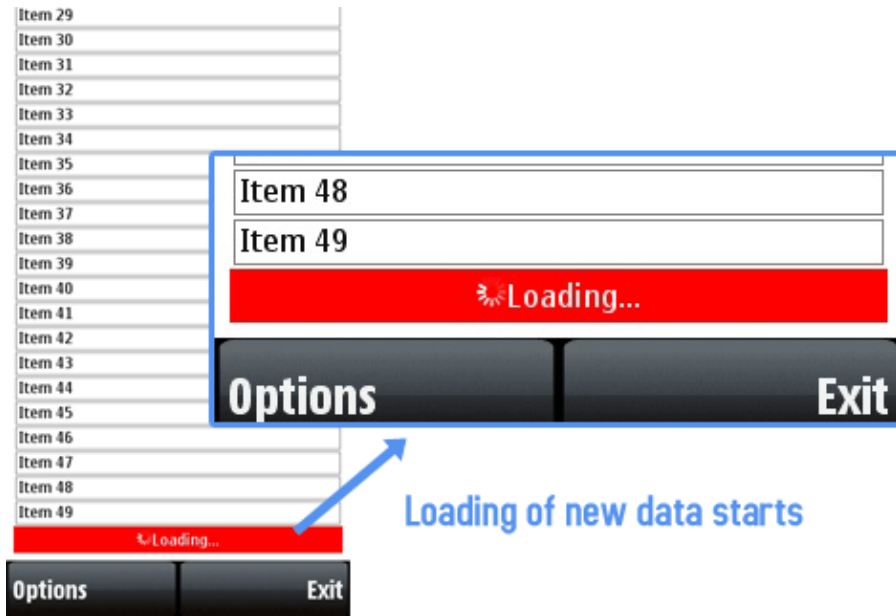
The **Live Scrolling** pattern will be implemented following these steps:

- an **HTML element** is defined to contain all the loaded data
- **the first set of data is loaded**, and shown inside the specified HTML element



- as soon as the **user scrolls to the bottom of the page**, the next set of data starts loading

How_to_implement_Live_Scrolling_in_a_Web_Runtime_widget



- when the new data is available, it is **appended to the existing one**



How to load remote data

The widget needs to **load data from a remote server**. This will be done by the following `loadXML()` file, structured as follows:

- creates an **XMLHttpRequest** object
- start the **asynchronous HTTP request** to retrieve the remote data
- if request goes well (`readyState = 4` and `HTTP status = 200`) then the **response XML is passed to the func() handler** passed as argument, otherwise the handler is called with **false** as argument

How_to_implement_Live_Scrolling_in_a_Web_Runtime_widget

```
function loadXML(xmlFile, func)
{
try
{
var request = new XMLHttpRequest();
    open("GET",xmlFile, true );
    send(null,request.

        onreadystatechange = function(){
if(this.readyState == 4)
{
if(this.status == 200)
{
    (request.responseXML);          func
}
else
{
    (false);          func
}
}
}
}
catch( e )
{
    (false);    func
}
}
```

Loading and displaying chunks of data

First, it's necessary to have an **HTML node where the data items will be appended**. The following **items_container** DIV will be used:

```
<body onload="init()">
<div id="items_container">
</div>
</body>
```

Now, to **load and display sets of data**, two functions are defined:

- **loadDataset(dataIndex)**: will load a specific set of data, by using the **loadXML()** function already defined
- **loadDatasetHandler()**: function that will **handle and display the loaded data**

```
function loadDataset (dataIndex)
{
    (loadXML/www.jappit.com/m/test/livescrolling/page.php?page=" + dataIndex, loadDatasetHandler)
}

function loadDatasetHandler (xml)
{
if (xml)
{
var items = xml.getElementsByTagName('item');
```

How_to_implement_Live_Scrolling_in_a_Web_Runtime_widget

```
for(var i = 0; i < items.length; i++)
{
    var itemElement = document.createElement('div');

    className = itemElement.className;

    innerHTML = itemElement.firstChild.nodeValue;

    appendChild(itemElement);
}
}
else
{
    alert("Error while loading page");
}
}
```

The **itemsContainer** variable is a global variable holding a **reference to the HTML container node** already defined, and is initialized in the widget's **init()** method, called by the **onload** event.

```
/* DOM element containing the data items */
var itemsContainer = null;

/* widget onload handler */
function init()
{
    itemsContainer = document.getElementById('items_container');
}
}
```

Notifying the user with a loading indicator

It's always good practice, when some **remote data is loading**, to give **visual information to the user with some sort of indicator**. This will be done by the following **showLoader()** and **hideLoader()** methods:

```
var loader = null;

function showLoader()
{
    if(loader == null)
    {
        loader = document.createElement('div');

        loader.id = 'loadingIndicator';

        var loaderImage = document.createElement('img');

        loaderImage.src = 'loadingImage/ajax-loader.gif';

        loader.appendChild(loaderImage);

        loader.appendChild(document.createTextNode('Loading...'));

        appendChild(loader);
    }
}

function hideLoader()
{
}
```

How_to_implement_Live_Scrolling_in_a_Web_Runtime_widget

```
if(loader != null)
{
    removeChild(loader);

    = null; loader
}
}
```

So, the load-related methods can be modified, in order to correctly show and hide the loading indicator.

```
function loadDataset (dataIndex)
{
    showLoader

    [...]
}
function loadDatasetHandler (xml)
{
    hideLoader

    [...]
}
```

Handling the scroll events

Fundamental part, in live scrolling, is to **check when the user reaches the end of page**, in order to start the loading of the next set of data. The **onscroll** window event will be used for this purpose.

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

    addEventListener("scroll", scrollHandler, false);
}

function scrollHandler()
{
    if(itemsContainer.offsetHeight + itemsContainer.offsetTop - document.body.scrollTop < screen.avai
    {
        ++; dataIndex

        (dataIndex); dataset
    }
}
```

What the **scrollHandler()** function does is to **check the vertical offsets of the DOM container**, comparing with the widget's total height and **current vertical scroll offset**. If the bottom part of the DOM element is shown, then the next set of data is loaded.

The **dataIndex** global variable holds the **index value of the last loaded data set**, in order to correctly load the next data set when required.

```
/* index of the last loaded data set */
var dataIndex = 0;
```

Loading the first set of data

Now, all is ready to **load the first set of data**. In order to do this, it is enough to call the `loadDataset()` method, with the starting dataset index (zero). The `loadDataset()` call is added to the `init()` method, after the already added code:

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

Downloads

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