How to update WRT widgets without installing a new version

This article explains **an alternative approach to update a Web Runtime widget**, that **does not require the installation of a new version** of the widget itself.

04 Oct 2010

Description

The **classical widget's update** mechanism, as for any other mobile applications, requires the **deployment and installation of newer versions** of the widget itself, when some changes in its code are required.

With Web Runtime widgets, and particularly with Web technologies, a new updating approach is possible.

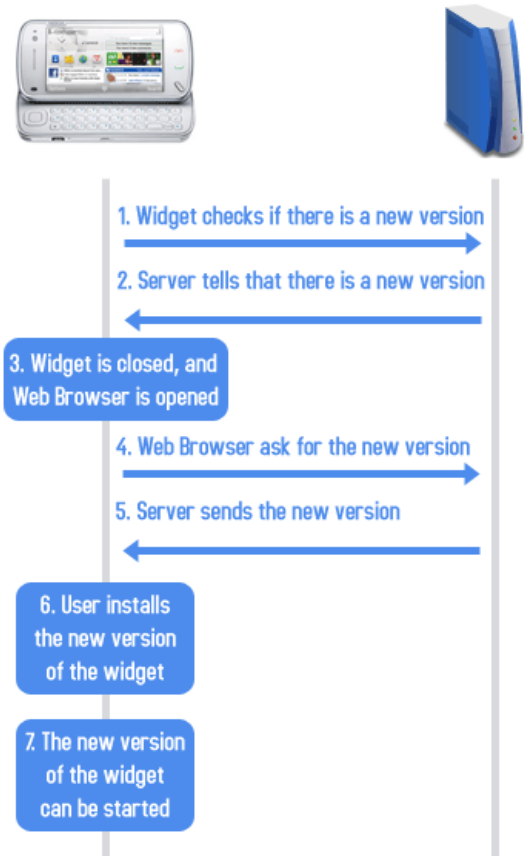
How it works

In order to update its own code, a Web Runtime widget must perform these steps:

1. **connect to a remote server**, to check if there is new code to be downloaded
2. if there is, the widget **downloads the new code**
3. if new code is downloaded, the widget **immediately uses it**
4. finally, the **new code is locally stored**, in order to be used for all the subsequent usage sessions, avoiding the need to download it each time
5. **at each subsequent start-up, the widget will load the locally stored JavaScript code**, saved from previous updates

Differences with classical Widgets' update

The classical Widget update process is shown by the following picture.

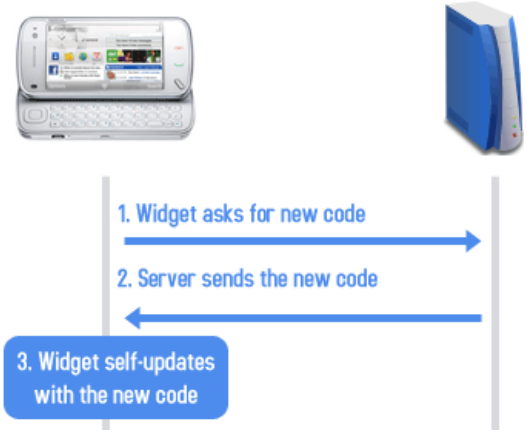


So, in order to update the widget, the user has to perform these actions:

1. authorize the widget to **connect to network**
2. authorize the widget to **open the device's Web browser**
3. authorize the **installation of the new widget**
4. **re-open the widget**

The approach proposed in this article, instead, is composed of very few steps, requires significantly fewer interactions from the user, as shown by the following picture, and so greatly improves the overall user experience:

(C) Copyright Nokia 2014. All rights reserved.



Only required user interaction, in this case, is the authorization to let the widget connect to network.

Also, this approach **optimizes the amount data sent through network**, and so reduces costs for the final users, reducing the transfer of data only to the modified portions of code.

Code example

A basic widget

First of all, let's define a very basic widget, whose purpose is to **alert the first and last name of a particular user**. The HTML body and CSS declarations are totally empty, so let's only see the initial **JavaScript code**:

```
var WIDGET_VERSION = "1.0";

var user = {
 'FirstName': 'Alessandro',
 'LastName': 'La Rosa'
};

function init()
{
 showUserInformation();
}

function showUserInformation()
{
 alert(user.LastName + " " + user.FirstName);
}

window.onload = init;
```

So, the **user** variable holds the data of the user, and the **showUserInformation()** alert this data, with the last name **before** the first name.

Now, let's imagine that the **showUserInformation() has to be changed**, in order to **alert the last name after the first name**. First thing to do, is to **request the new code from a remote server**.

The **WIDGET_VERSION** variable stores the value of the current version of the widget, and it will be used when requesting updates to the remote server.

Requesting the new code

The following **httpRequest()** method is an utility method that **performs an XMLHttpRequest** to the URL passed as argument, and then **calls a callback handler**, passed as second argument, passing the response text (or false in case of errors).

```
function httpRequest(url, handler)
{
```


(C) Copyright Nokia 2014. All rights reserved.

```
 try
 {
  var request = new XMLHttpRequest();
  request.open("GET", url, true );
  request.send(null);

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

So, it is now possible to request the new JavaScript code from the remote server:

```
function requestNewCode()
{
 httpRequest("http://www.jappit.com/m/forumnokia/widget_updates.php?version=" +
WIDGET_VERSION, newCodeHandler);
}
function newCodeHandler(responseText)
{
 if(responseText)
 {
  if(responseText != 'NO UPDATES')
  {
   //use the new code!
  }
 }
 else
 {
  alert("Error while requesting the updates");
 }
}
```

The **requestNewCode()** functions just calls the **httpRequest** method to perform an HTTP request to the remote server. When this request is complete, the **newCodeHandler()** gets called, with the response text passed as parameter. If some text is returned from the server side, then this means that new code has been sent, and the widget needs to handle it.

Before dealing with the handling of the new code sent from the server, let's see a sample server script that will handle the code updates.

The remote server

(C) Copyright Nokia 2014. All rights reserved.

The server side **must send new code only to older versions of the widget**, so it needs to check the **version** parameters sent from the widget itself. A **sample PHP script**, serving code to this example widget, is the following:

```
<?

if($_REQUEST['version'] == "1.0")
{
 echo
  "function showUserInformation(){" .
   "alert(user.FirstName + ' ' + user.LastName);" .
  "}" .
  "WIDGET_VERSION = '1.1'";
}
else
{
 echo 'NO UPDATES';
}

?>
```

What the server side script does is:

- sending a **modified version of the showUserInformation()** function
- sending the **new widget version number**

Handling the new JavaScript code

When the server recognize a widget as old, it sends only the strictly necessary JavaScript code in order to update it. So, when the widget receives it, it has to do 2 things:

- immediately **use the new code**
- **store it, for subsequent usage sessions**

Use the new code

First, let's see how to use the code sent from the remote server:

```
function addNewCode(code)
{
 var scriptElement = document.createElement('script');

 var scriptContent = document.createTextNode(code);

 scriptElement.appendChild(scriptContent);

 document.body.appendChild(scriptElement);
}
```

What the **addNewCode()** function does is:

- create a new <script> DOM element
- append the JavaScript code inside this element
- append the newly created <script> element to the widget's body

This results in 2 things:

- the **new code gets immediately executed**
- **functions and variables** already defined with the same name **gets overridden**, since the new code is appended just at the end of the HTML document.

Save the code for future usage sessions

(C) Copyright Nokia 2014. All rights reserved.

So far, the widget has downloaded and used the new JavaScript code, and so it is up to date. But, **how to maintain the updates once the widget is restarted?**

Web Runtime widgets allow to save and load local data via the **widget's setPreferenceForKey() and preferenceForKey() methods**. So, the widget just needs to use these methods in order to store the updates, and to use them once it is restarted.

The following **storeNewCode()** methods will store the new JavaScript code, maintaining the code already potentially saved in previous updates:

```
function storeNewCode(code)
{
 var alreadySavedCode = widget.preferenceForKey('code_updates');

 if(alreadySavedCode != undefined)
  widget.setPreferenceForKey(alreadySavedCode + code, 'code_updates');
 else
  widget.setPreferenceForKey(code, 'code_updates');
}
```

So, the **newCodeHandler()**, defined above, can be modified as follows:

```
function newCodeHandler(responseText)
{
 if(responseText)
 {
  if(responseText != 'NO UPDATES')
  {
   alert("there is new code: " + responseText);

   addNewCode(responseText);

   storeNewCode(responseText);
  }
  else
  {
   alert("no updates available");
  }
 }
 else
 {
  alert("Error while requesting the updates");
 }
}
```

Use the code already saved

Once the widget has been updated with new code, and this code has been locally saved, the widget needs to use it in all its subsequent usage sessions. In order to do this, the **loadLocalUpdates()** method is defined:

```
function loadLocalUpdates()
{
 var localUpdates = widget.preferenceForKey('code_updates');

 if(localUpdates != undefined)
  addNewCode(localUpdates);
}
```


(C) Copyright Nokia 2014. All rights reserved.

Since this should be the first thing that the widget does after having loaded, let's modify the previously defined **init()** function:

```
function init()
{
 loadLocalUpdates();

 showUserInformation();
}
```

Advantages

The approach described in this articles has the following advantages over the classical update mechanism:

- **it does not require a full new version of the widget to be deployed and installed**. This improves the final **user experience** for both experienced and non-experienced users, since both of them will not require to perform a full installation process, only requiring a simple network request to be performed.
- **When updating** a widget with a new version, **all the data stored from the older version gets lost**. So, the new widget cannot access it anymore. **With this alternative approach**, since there's no "real" update of the widget, **all the data remains stored, and newer versions can continue to use it as well**.

Limits

Biggest limit of this approach is that current Web Runtime APIs allow to save **only textual data**. This means that, when also **binary data (e.g.: images) needs to be added or updated, this approach will not work**, and so a classical update is required in order to deploy the new resources.

When to use it

In general, this approach can be used whenever the JavaScript code of a Web Runtime widget needs to be updated. Since it does not require all the steps of a classical application update, it **enhances user experience**, letting your users always have the latest updates and functionality with a **smooth, integrated and transparent update mechanism**. Also, since updates are totally transparent to the user, this approach **gives to developers the opportunity to release code updates more often** than with classical updates, so further improving the quality of their widgets.

Specific usage **scenarios** include:

- **Bug fixes and code changes**: code can be easily replaced, and older "bugged" functions can be redefined by newer code
- Whole **new features**: new features can be easily added, just modifying the existing code in order to call the new features

Downloads

The sample widget, implemented in this article, is available for download here: Media:SelfUpdatableWidget.zip

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.