Implementing widget runtime APIs with Nokia WRT functionality

This article is based on my personal experiences of writing wrappers for Opera widgets

Preface

In a perfect world, a widget would run equally well on all of the different runtimes. However, as there is not yet an established standard for the services a widget engine should provide, or how a widget should be structured and packaged, compatibility issues still exist. This means that widgets need to be ported from one environment to another, and that Javascript wrappers are needed to provide a layer of interpretation for the similar functionality of a foreign environment, so that a widget can use the APIs it expects to be able to use.

The following is a general guide for writing Javascript wrappers to mimic runtime specific APIs with Nokia WRT functionality.

Overview

In general, there are 3 main steps involved in creating a Javascript wrapper.

1. Create a Javascript file that can be included in the widget being ported.
2. Create a Javascript object that has the same members as the API you wish to implement.
3. Initialize the values of the attributes.
4. Implement the methods using Nokia WRT functionality.

Create a Javascript Object with the required slots

The clearest way to create the object you wish to use to implement the API you are porting is to use Javascript Object Notation (JSON). If our API had 2 attributes and 2 methods, we could define it as follows.

```
var myWidgetAPI = {

        // Widget API attributes

 "attribute1": null, // initialized after object is created
 "attribute2": null, // initialized after object is created

 // Widget API methods

 "method1": myMethod1,
 "method2": myMethod2

}
```

Initialize the values of attributes

The attributes you wish to initialize may be directly accessible, or may need to be interpreted from other information available from Nokia WRT

Static attributes

These are attributes that do not change while the widget is running. An example would the originURL of a widget (the path to the configuration file of the widget) or its identifier.

These may be directly available from Nokia WRT, and simply need to be assigned to the slot. Other times, the information is a little more hidden and needs to be found using one of the service APIs provided by Nokia WRT. Once the information has been found and possibly modified, it can be assign to the appropriate member slot of the object you have created.

(C) Copyright Nokia 2014. All rights reserved.

Dynamic attributes

These are attribute that can change while the widget is running. An example would be the widget mode, whether it is docked or fullscreen etc. These are more difficult to implement as they need to be either updated when the method is called, updated when an event is fired indicating a state change, or continuously updated using polling.

Implement the methods

The ability to implement methods depends on whether or not there is similar functionality available from within Nokia WRT. Sometimes the method will function exactly the same, other times it will work similarly and other times not at all. To implement a method slightly differently or not at all is a judgement call, depending on whether the benefits of implementing it are worth the side effects of it not having exactly the same behaviour as is expected.

Implementing Synchronous Methods

Synchronous methods can be implemented by assigning a function reference to the appropriate slot of the Javascript object and then later defining the function in your wrapper.

A simple example would be implementing the Opera Widget hide() method. Identical behaviour can be achieved within Nokia WRT by calling the homescreen (UID = 0x102750F0) to the foreground, essentially sending the widget to the background.

```
var myWidgetAPI = {
    "hide" = hide
}

function hide()
{
    widget.openApplication(0x102750F0);
}
```

Implementing Asynchronous Methods

Asynchronous methods are a little more complicated to implement. The following is a basic workflow:

1. Define the function so that it accepts the same arguments as the method you are wrapping
2. Copy the callback function reference to a global variable so that it can be later used.
3. Call the appropriate Nokia WRT asynchronous method, passing appropriate arguments, using your own intermediate callback method to intercept the returned result.
4. Define the intermediate callback function to intercept the returned result from Nokia WRT.
5. Reformat the result so that it resembles what should be returned by the wrapper.
6. Call the callback function that was originally stored to a global variable, passing the reformatted return result as an argument.

Examples

Here are two wrappers that illustrate one method of providing the needed compatibility layer between the ported widget and the Nokia WRT environment.

- OperaWidget.txt
- OperaGeolocationWrapper.txt

Remember that to use these wrappers, they need to be renamed and included in your widget. For more information about porting widgets, refer to the following articles.

- How to port Bondi Widgets to Symbian Web Runtime
- How to port Opera Widgets to Nokia WRT

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.