Extending Bondi wrapper library

This article is based on my personal experiences of writing wrappers for Bondi 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 guide for extending the Bondi JavaScript wrapper library to mimic runtime specific APIs with Nokia WRT functionality. This guide is part of How to port Bondi Widgets to Nokia WRT instructions.

Currently supported methods

- bondi.devicestatus.getPropertyValue({aspect:Battery,property:batteryBeingCharged})
- bondi.geolocation.getCurrentPosition()

The wrapper logic

Bondi JavaScript wrapper library was developed using JSON (JavaScript Object Notation). The easiest way to extend the library is to study the existing wrapper functions and mimic them.

The original Bondi API call to get battery level is:

```
var ref = {};
ref.vocabulary = "http://bondi.omtp.org";
ref.aspect = "Battery";
ref.property = "batteryBeingCharged";
var value = bondi.devicestatus.getPropertyValue(ref);
```

The `getPropertyValue()` function takes aspect, property and vocabulary parameters to define which property value it returns from the devicestatus API.

The Nokia WRT equivalent call to get battery level is:

```
var serviceObj = device.getServiceObject("Service.SysInfo", "ISysInfo");
var criteria = {};
criteria.Entity = "Battery";
criteria.Key = "ChargingStatus";
var result = serviceObj.ISysInfo.GetInfo(criteria);
val value = result.ReturnValue.Status;
```

The service object is defined as a System Info object, which can be considered the equivalent of Bondi's devicestatus.

`criteria.Entity` is the equivalent of `ref.aspect` and `criteria.Key` is the equivalent of `ref.property`.

There is a difference between the returned value in Bondi and Nokia WRT. Bondi's battery charging status is stored directly in the `value` variable, whereas in Nokia WRT the actual battery charging status is stored in the `result.ReturnValue.Status`.

With these differences taken into consideration, we can now create an object that follows the Bondi structure, but uses the Nokia WRT methods:

```
var bondi = {
 devicestatus: {
  getPropertyValue: function (ref) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
    try {
     serviceObj = device.getServiceObject("Service.SysInfo", "ISysInfo");
    } catch (ex) {
     alert("Service object cannot be found.");
    }

    var criteria = new Object();

    if(ref.aspect == "Battery"){
     criteria.Entity = "Battery";
     if (ref.property == "batteryBeingCharged"){
      criteria.Key = "ChargingStatus";
      try {
       var result = serviceObj.ISysInfo.GetInfo(criteria);
       return result.ReturnValue.Status;
      } catch (ex) {
       alert("Error getting charging status: " + ex);
      }
     }
    }
   }
  }
 }
```

With this wrapper, the original Bondi function can be used to retrieve the battery charging status of a Nokia device.

Extending the library

Synchronous functions

Synchronous functions are pretty straightforward to wrap. The following is a basic work-flow for synchronous functions:

1. Create the Bondi object structure
2. Define the function so that it accepts the same arguments as the method you are wrapping
3. Implement the Nokia WRT equivalent methods
4. Return the resulting value according to the Bondi specification

The battery charging status wrapper example uses synchronous function calls on both Bondi and Nokia WRT APIs.

Asynchronous functions

Asynchronous functions are a little more complicated to implement. The following is a basic work-flow for asynchronous functions:

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

The geolocation part of the wrapper uses asynchronous functions. The Bondi geolocation API is based on W3C standards, whereas the Nokia WRT equivalent is not. Here is where the Platformservices JavaScript wrapper library comes in. Platformservices wraps the Nokia WRT geolocation API to follow the W3C standards, which means that the Bondi and Nokia WRT geolocation functions look and behave the same way. Therefore the original Bondi callback functions can be used directly with the Nokia WRT geolocation function.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.