Windows Phone 8 - Implementing Apache Cordova plugin

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/adding-native-functionality/windows-phone-8-adding-native-functionality-hybrid-application-apache-cordova-plugin.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

In some cases, developers of a MobileFirst application might have to use a specific third-party native library or a device function that is not yet available in Apache Cordova.

With Apache Cordova, developers can create an Apache Cordova plug-in, which means that they create custom native code blocks, and call these code blocks in their applications by using JavaScript.

This tutorial demonstrates how to create and integrate a simple Apache Cordova plug-in for Windows Phone 8, in the following topics:

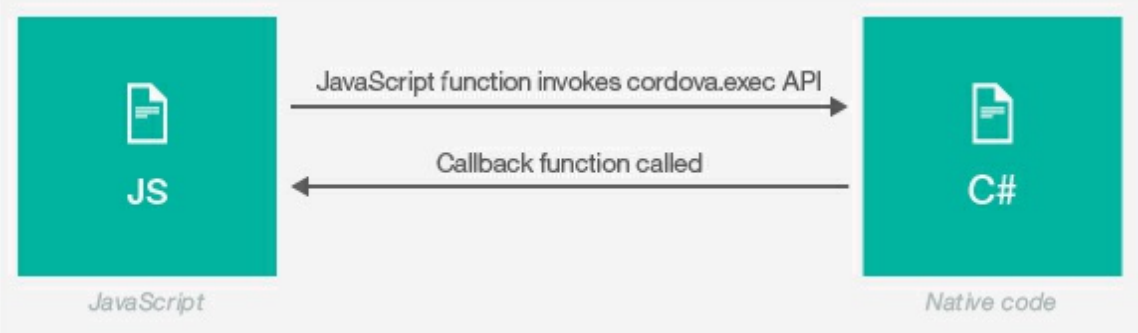
- Creating a plug-in
- Declaring a plug-in
- Implementing cordova.exec() in JavaScript
- Implementing the C# code of a Cordova plug-in
- Sample application

Note: In Cordova-based applications, developers must check for the `deviceready` event before they use the Cordova API set. In a MobileFirst application, however, this check is done internally.

Instead of implementing this check, you can place implementation code in the `wlCommonInit()` function in the `common\js\main.js` file.

Creating a plug-in

1. Declare the plug-in in the `config.xml` file.
2. Use the `cordova.exec()` API in the JavaScript code.
3. Create the plug-in class that will run natively in Windows Phone 8. The plug-in performs the required action and calls a JavaScript callback method that is specified during the call to `cordova.exec()`



Declaring a plug-in

You must declare the plug-in in the project, so that Cordova can detect it. To declare the plug-in, add a reference to the `config.xml` file, located in the native folder of the Windows Phone 8 environment.

```
<feature name="sayHelloPlugin">
   <param name="wp-package" value="sayHelloPlugin" /
>
</feature>
```

Implementing cordova.exec() in JavaScript

From the JavaScript code of the application, use the `cordova.exec()` method to call the Cordova plug-in:

```
function sayHello() {
   var name = $("#NameInput").val();
   cordova.exe(sayHelloSuccess, sayHelloFailure, "SayHelloPlugin", "sayHello", [name])
;
}
```

- `sayHelloSuccess` - Success callback
- `sayHelloFailure` - Failure callback
- `SayHelloPlugin` - Plug-in name as declared in the `config.xml` file
- `sayHello` - Action name
- `[name]` - Parameters array

The plug-in calls the `success` and `failure` callbacks.

```
function sayHelloSuccess(data){
   WL.SimpleDialog.show(
      "Response from plug-in",
      data,
      [{text: "OK", handler: function() {WL.Logger.debug("Ok button pressed");}}
]
   );
}

function sayHelloFailure(data){
   WL.SimpleDialog.show(
     "Response from plug-in",
     data,
     [{text: "OK", handler: function() {WL.Logger.debug("Ok button pressed");}}]
   );
}
```

Implementing the C# code of a Cordova plug-in

After you have declared the plug-in and the JavaScript implementation is ready, you can implement the Cordova plug-in. For this purpose, ensure that the project is built in Eclipse and opened in the Visual Studio IDE.

Step 1

1. Create a new C# class.
2. Add the new class to your project namespace and add the required import statements.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using WPCordovaClassLib.Cordova;
using WPCordovaClassLib.Cordova.Commands;
using WPCordovaClassLib.Cordova.JSON;

namespace Cordova.Extension.Commands {
   public class SayHelloPlugin : BaseCommand
{
```

Step 2

Implement the `SayHelloPlugin` class and the `sayHello` method.

1. The JavaScript wrapper calls the `sayHello` method and passes a single parameter. It returns a string back to JavaScript.

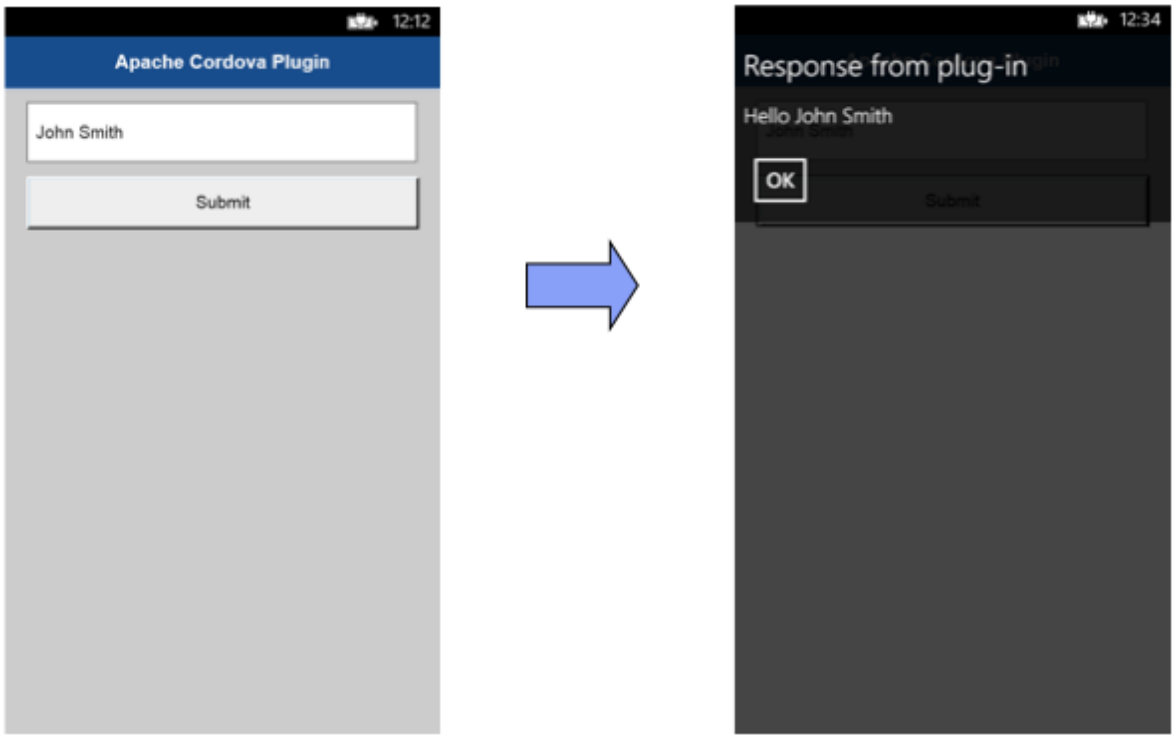
```
        public void sayHello(string options) {
            string optVal = null;

            try {
                optVal = JsonHelper.Deserialize<string[]>(options)[0];
            }
            catch (Exception) {
                DispatchCommandResult(new PluginResult(PluginResult.Status.ERROR, "SayHelloPlugin
signaled an error"));
            }
```

2. The `DispatchCommandResult` method returns the result to JavaScript, whether success or failure.

```
            if (optVal == null) {
                DispatchCommandResult(new PluginResult(PluginResult.Status.ERROR, "Got null value
as input"));
            } else {
                DispatchCommandResult(new PluginResult(PluginResult.Status.OK, "Hello " + optVal));
            }
        }
    }
}
```

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/ApacheCordovaPluginsProject.zip) the Studio project.


12:12
Apache Cordova Plugin
John Smith
Submit
12:34
Response from plug-in
Hello John Smith
OK
Submit