Android - Implementing Apache Cordova plugin

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.

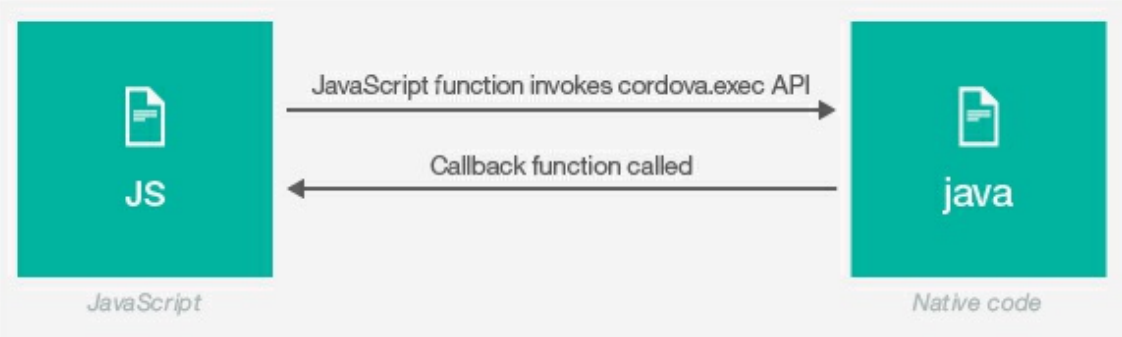
Note: In Cordova-based applications, developers must check for the `deviceready` event before they use the Cordova API set. In a MobileFirst application, however, this check is done internally.

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

Jump to:

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 Android. 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\res\xml` folder in the Android environment.

```
<feature name="sayHelloPlugin">
    <param name="android-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.exec(sayHelloSuccess, sayHelloFailure, "SayHelloPlugin", "sayHello", [nam
e]);
}
```

`sayHelloSuccess` - Success callback `sayHelloFailure` - Failure callback `SayHelloPlugin` - Plug-in name as declared in `config.xml` `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 Java 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.

1. Add a new Java class file.
2. Extend the `org.apache.cordova.CordovaPlugin` class and add the required import statements.

   ```
   public class SayHelloPlugin extends CordovaPlugin {
   ```

3. Implement an `execute` method.
   - The arguments contain information that is required by a plug-in, such as action, arguments array, and callback context.

     ```
     public boolean execute(String action, JSONArray args, CallbackContext ca
     llbackContext)
         throws JSONException {
     ```

4. If the supplied action is `sayHello`, retrieve the first argument from the `args` array, prepare a `responseText` string and, by using the `callbackContext` argument, call the `success` callback with this `responseText` string as the argument.

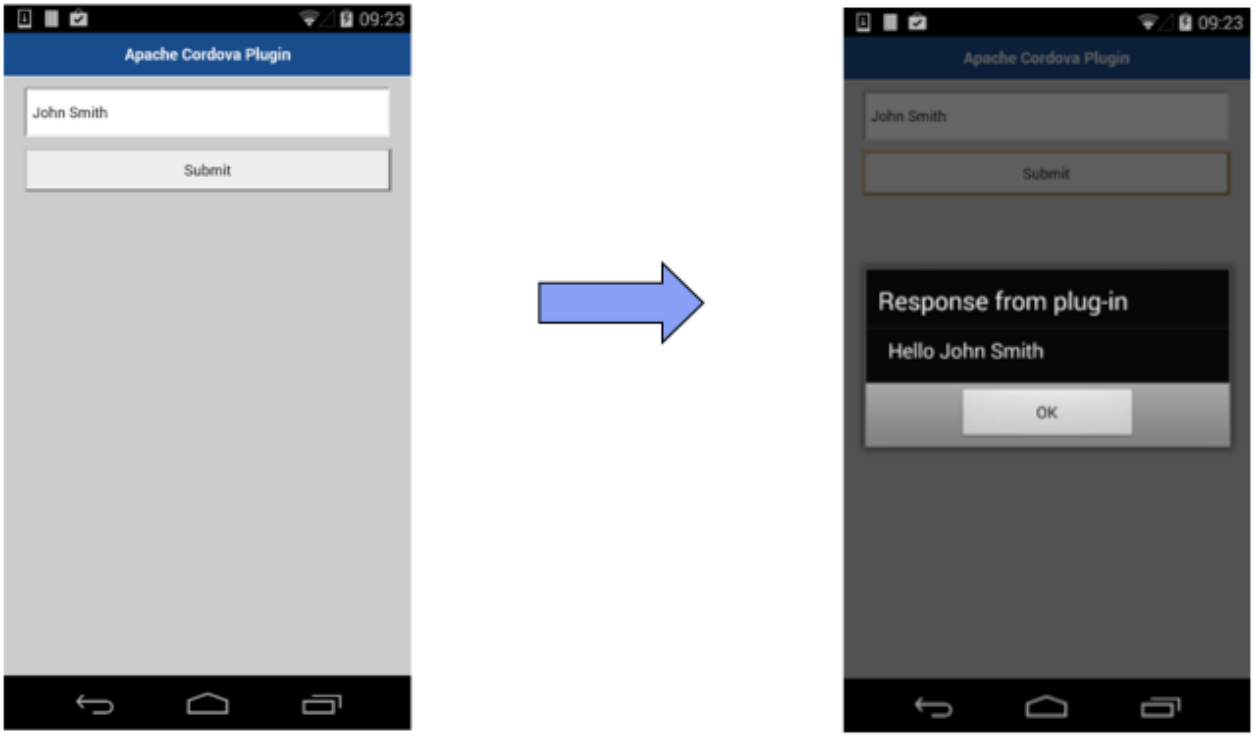
   ```
   if (action.equals("sayHello")){
       try {
           String responseText = "Hello " + args.getString(0);
           callbackContext.success(responseText);
       } catch (JSONException e){
           callbackContext.error("Failed to parse parameters");
       }
       return true;
   }
   ```

5. Returning `false` means that the action that is supplied from JavaScript was not recognized.

```
    return false;
}
```

Sample application

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



Last modified on