# Android - Implementing Cordova plug-ins

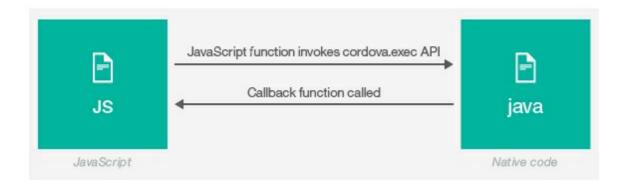
#### **Overview**

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. In this tutorial, a simple Apache Cordova plug-in creation and integration for Android will be demonstrated. **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, implementation code can be placed in the wlCommonInit() function in commonjsmain.js.

The below code blocks are based on the sample application, provided at the bottom of this tutorial.

### Plug-in creation overview:

- Declare the plug-in in the config.xml file
- Use the cordova.exec() API in the JavaScript code
- 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

The plug-in needs to be declared 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 nativeresxml 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 cordova.exec() to call the Cordova plug-in:

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

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 the plug-in is declared, and the JavaScript implementation is ready, the Cordova plug-in can be implemented.

### Step 1

- · Add a new Java class file
- Extend the org.apache.cordova.CordovaPlugin class and add the required import statements

```
\textbf{public class SayHelloPlugin extends} \ \mathsf{CordovaPlugin} \ \{
```

### Step 2

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 callbackContext)
throws JSONException {
```

• If the supplied action is sayHello, retrieve the first argument from the args array, prepares a responseText string and, by using the callbackContext argument, calls 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;
}
```

• 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/v630/ApacheCordovaPluginsProject.zip) the Studio project.





