

# iOS - 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.

This tutorial demonstrates a simple Apache Cordova plug-in creation and integration for iOS.

**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.

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

### Plug-in creation overview:

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 iOS.

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 native folder of the iOS environment.

```

1 <feature name="sayHelloPlugin">
2   <param name="ios-package" value="sayHelloPlugin" />
3 </feature>
  
```

## Implementing `cordova.exec()` in JavaScript

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

```

1  function sayHello() {
2      var name = $("#NameInput").val();
3      cordova.exec(sayHelloSuccess, sayHelloFailure, "SayHelloPlugin", "sayHello", [name]);
4  }

```

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.

```

1  function sayHelloSuccess(data){
2      WL.SimpleDialog.show(
3          "Response from plug-in", data,
4      [{text: "OK", handler: function() {WL.Logger.debug("Ok button pressed");}}]
5      );
6  }
7  function sayHelloFailure(data){
8      WL.SimpleDialog.show(
9          "Response from plug-in", data,
10     [{text: "OK", handler: function() {WL.Logger.debug("Ok button pressed");}}]
11     );
12 }

```

## Implementing the Objective-C 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. For this purpose, ensure that the project is built in Eclipse and opened in the Xcode IDE.

### Step 1

1. Add a new Cocoa Touch Class file, make sure that it is a subclass of `UIViewController`, and save it in the `Classes` folder of the Xcode project.
2. Import the `Cordova/CDV.h` and inherit the `CDVPlugin` class.
3. Declare the `SayHelloPlugin` signature.

```

1  #import <Foundation/Foundation.h>
2  #import <Cordova/CDV.h>
3  <p>@interface SayHelloPlugin : CDVPlugin
4  - (void)sayHello:(CDVInvokedUrlCommand*)command;
5  @end

```

### Step 2

- Implement the method. The `command` argument contains references to the parameters that are sent from JavaScript and callbacks:

```

1  #import "SayHelloPlugin.h"
2  @implementation SayHelloPlugin
3  - (void)sayHello:(CDVInvokedUrlCommand*)command {

```

- This statement retrieves the parameters that are sent from JavaScript.

```
1 | NSString *responseString = [NSString stringWithFormat:@"Hello %@", [command.arguments objectAtIndex:0]];
```

- The `pluginResult` object is created with data retrieved from JavaScript. The `CDVCommandStatus` parameter defines whether the plug-in call was successful or not.

```
1 | CDVPluginResult *pluginResult = [CDVPluginResult resultWithStatus:CDVCommandStatus_OK messageAsSuccess:[responseString UTF8String]];
2 |
```

- The `sendPluginResult` method is used to return a response back to JavaScript (invoke callback).

```
1 | [self.commandDelegate sendPluginResult:pluginResult callbackId:command.callbackId];
2 | }
3 | @end
```

### Important:

If you are working with existing `.m` and `.h` files, reference those files while you are working in Xcode.

Placing the `.m` and `.h` files only in the `iphone\native\Classes` folder in Eclipse is not sufficient, because these files will not be referenced in the Xcode project unless you add them in Xcode.

## Sample application

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/ApacheCordovaPluginsProject.zip>)  
the Studio project.



