

# Invoking adapter procedures from native iOS Swift applications

fork and edit tutorial (<https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/>) | report issue (<https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/issues/new>)

## Overview

To create and configure an iOS native project, first follow the "Creating your first Native iOS MobileFirst application ([../hello-world/creating-first-native-ios-mobilefirst-application/](https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/blob/master/hello-world/creating-first-native-ios-mobilefirst-application/))" tutorial.

Make sure that you follow the extra steps for Swift-based applications.

## Initializing WLClient

1. Access the `WLClient` functionality by calling the `WLClient.sharedInstance` method anywhere in your application.
2. Initiate the connection to the server by using the `wlConnectWithDelegate` method.  
For most actions, you must specify a delegate object, such as a `MyConnectListener` instance in the following example:

```
let connectListener = MyConnectListener(vc: self)
WLClient.sharedInstance().wlConnectWithDelegate(connectListener)
```

3. Make sure that your Bridging Header includes `WLSwiftBridgingHeader.h` for access to MobileFirst APIs.
4. To specify the delegate object, create a delegate for the `wlConnectWithDelegate` method and receive the response from the MobileFirst Server instance. Name the class `MyConnectListener`. For your `MyConnectListener` class, the header file must specify that it implements the `WLDelegate` protocol.  
**Note:** To avoid a compiler error raising that your delegate does not conform to `NSObjectProtocol`, make your class a subclass of `NSObject`.

```
class MyConnectListener: NSObject, WLDelegate{
    //...
}
```

The `WLDelegate` protocol specifies that the class implements the following methods:

- The **onSuccess** method: `func onSuccess(response: WLResponse!)`
- The **onFailure** method: `func onFailure(response: WLFailResponse!)`

After `wlConnectWithDelegate` finishes, the `onSuccess` method or the `onFailure` method of the supplied `MyConnectListener` instance is called.

In both cases, the response object is sent as an argument.

5. Use this object to operate data that is retrieved from the server.

```

func onSuccess(response: WLResponse!) {
    var resultText = "Connection success. "
    if(response != nil){
        resultText += response.responseText
    }
    self.vc.updateView(resultText)
}
func onFailure(response: WLFailResponse!) {
    var resultText = "Connection failure. "
    if(response != nil){
        resultText += response.errorMsg
    }
    self.vc.updateView(resultText)
}

```

## Calling an adapter procedure

1. To call a procedure, create a `WLProcedureInvocationData` object and specify the adapter name and the procedure name.

```

let invocationData = WLProcedureInvocationData(adapterName: "RSSReader", procedureName: "getStories")

```

2. Call the procedure by using the shared instance of `WLClient`. As previously stated, supply a delegate object to manage the retrieved data.

```

let invokeListener = MyInvokeListener(vc: self)
WLClient.sharedInstance().invokeProcedure(invocationData, withDelegate: invokeListener)

```

## Receiving a procedure response

When the procedure call is complete, a delegate method of the `MyInvokeListener` instance is called. Any delegate header file must specify that it complies with a `WLDelegate` protocol.

```

class MyInvokeListener: NSObject, WLDelegate{
}

```

After the procedure call finishes, the `onSuccess` method or the `onFailure` method of the supplied `MyInvokeListener` instance is called.

In both cases, a response object is sent as an argument.

3. Use this object to operate data that is retrieved from the server.

```

func onSuccess(response: WLResponse!) {
    var resultText = "Invocation success. "
    if(response != nil){
        resultText += response.responseText
    }
    self.vc.updateView(resultText)
}
func onFailure(response: WLFailResponse!) {
    var resultText = "Invocation failure. "
    if(response != nil){
        resultText += response.errorMsg
    }
    self.vc.updateView(resultText)
}

```

## Sample application

The attached sample contains two projects:

- The **InvokingAdapterProceduresNativeProject.zip** file contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The **InvokingAdapterProceduresSwiftProject.zip** file contains a native iOS Swift application that uses a MobileFirst native API library to communicate with the MobileFirst Server instance.

Make sure to update the **worklight.plist** file in **SwiftNativeApp** with the relevant server settings.

Click to download

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

Click to download

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



Invocation Success:

/^secure-

```
{ "responseID": "10", "statusCode": 200, "errors":
```

```
[], "isSuccessful": true, "statusReason": "OK", "rss":
```

```
{ "dc": "http://purl.org/Vdc/elementsV1.1V", "itunes": "http://www.itunes.com/VdtdsVpodcast-1.0.dtd", "channel":
```

```
{ "title": "Engadget", "description": "Engadget", "item": { "guid":
```

```
{ "CDATA": "http://www.engadget.com/V2012/V11/V04/microsoft-posts-build-2012-session-videos-for-eager-windows-8-
```

```
devsV", "isPermaLink": "true"}, "pubDate": "Sun, 04 Nov 2012 03:39:00
```

```
EDT", "category": ["azure", "build 2012", "Build2012", "developer", "html5", "
```