Invoking adapter procedures from native iOS Swift applications

Overview

To create and configure an iOS native project, first follow the “Configuring a native iOS application with the MobileFirst Platform SDK (../../hello-world/configuring-a-native-ios-with-the-mfp-sdk/)” tutorial.

If you are developing Swift-based applications, make sure that you follow the additional steps.

MobileFirst applications can adapt procedures to communicate with any data source. This tutorial explains how to use the REST API for returning data from an HTTP adapter. The same can be applied using other data sources (such as SQL adapters, etc).

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:

   ```
   <br />
   let connectListener = MyConnectListener(vc: self)<br />
   WLClient.sharedInstance().wlConnectWithDelegate(connectListener)<br />
   ```

3. Make sure that your Bridging Header includes the `WLSwiftBridgingHeader.h` file 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 passed 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

The `WLResourceRequest` class handles resource requests to MobileFirst adapters or external resources.

1. To call a procedure, create a `WLResourceRequest` object and specify the path to the adapter and the HTTP method.

```
let request = WLResourceRequest(URL: NSURL(string: "/adapters/RSSReader/getStories"), method: WLHttpMethodGet)
```

2. Add the required parameters.
   - For JavaScript-based adapters, use the `params` parameter name to set an array of parameters.

```
request.setQueryParameterValue("['technology']", forName: "params")
```

   - For Java adapters or other resources, you can use `setQueryParameterValue` for each parameter.

```
request.setQueryParameterValue("value1", forName: "param1")
request.setQueryParameterValue("value2", forName: "param2")
```

3. Call the procedure by using the `sendWithCompletionHandler` method. Supply a completion handler to manage the retrieved data.

```
request.sendWithCompletionHandler { (WLResponse response, NSError error) -> Void in
    var resultText = ""
    if(error != nil){
        resultText = "Invocation failure. "
        resultText += error.description
    }
    else if(response != nil){
        resultText = "Invocation success. "
        resultText += response.responseText
    }
    self.updateView(resultText)
}
```

For more granular management of the retrieved data (such as non-text responses, PDF, etc), you can use the `sendWithDelegate` method and provide a delegate which conforms to both the `NSURLConnectionDataDelegate` and `NSURLConnectionDelegate` protocols.

Other signatures, which are not covered in this tutorial, exist for the `send` method. Those will enable you to set parameters in the body instead of the query, or handle the response with a delegate instead of a completion handler. See the user documentation to learn more.

> Learn more about `WLResourceRequest` in the user documentation.

Sample application

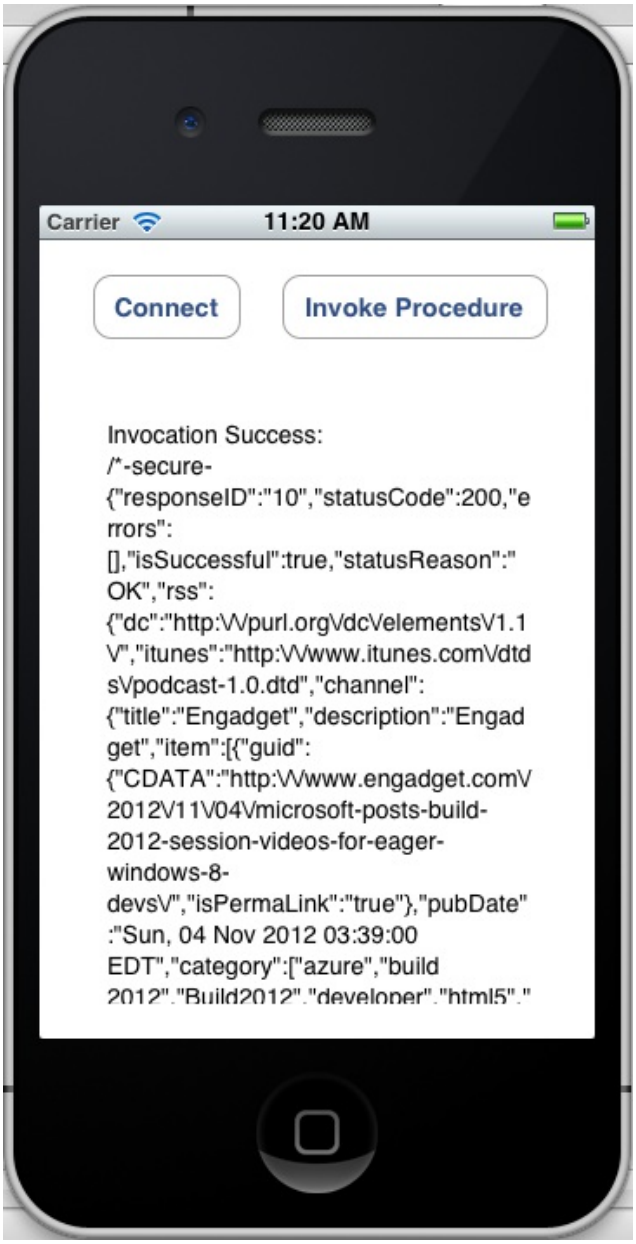
Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/InvokingAdapterProceduresNativeProject.zip) the Studio project.

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

The sample contains two projects:

- The `InvokingAdapterProceduresNativeProject.zip` file contains a **MobileFirst native API** that you can deploy to your MobileFirst Server instance.
- 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.


Carrier
11:20 AM
Connect
Invoke Procedure
Invocation Success:
/*-secure-
{"responseID":"10","statusCode":200,"errors":[],"isSuccessful":true,"statusReason":"OK","rss":{"dc":"http:\/\/purl.org\/dc\/elements\/1.1\/","itunes":"http:\/\/www.itunes.com\/dtds\/podcast-1.0.dtd","channel":{"title":"Engadget","description":"Engadget","item":[{"guid":{"CDATA":"http:\/\/www.engadget.com\/2012\/11\/04\/microsoft-posts-build-2012-session-videos-for-eager-windows-8-devs\/","isPermaLink":"true"},"pubDate":"Sun, 04 Nov 2012 03:39:00 EDT","category":["azure","build 2012","Build2012","developer","html5",