

Java HTTP Adapter

Overview

This tutorial is a continuation of Java Adapter ([../../server-side-development/java-adapter/](http://www.ibm.com/developerworks/mobilefirst/server-side-development/java-adapter/)) and assumes previous knowledge of the concepts described there.

Java adapters provide free reign over connectivity to your backend. It is therefore your responsibility to ensure best practices regarding performance and other implementation details.

This tutorial shows an example of a Java adapter that connects to an RSS feed by using a Java `HttpClient`.

Topics:

- `RSSAdapterApplication`
- `RSSAdapterResource`
- Results

RSSAdapterApplication

`RSSAdapterApplication` extends `MFPJAXRSApplication` and is a good place to trigger any initialization required by your application.

```
@Override
protected void init() throws Exception {
    RSSAdapterResource.init();
    logger.info("Adapter initialized!");
}
```

RSSAdapterResource

```
@Path("/")
public class RSSAdapterResource {
}
```

`RSSAdapterResource` is where we handle the requests to your adapter.

`@Path("/")` means that the resources will be available at the URL `http(s)://host:port/ProjectName/adapters/AdapterName/`.

HTTP Client

```

private static CloseableHttpClient client;
private static HttpHost host;
public static void init() {
    client = HttpClients.createDefault();
    host = new HttpHost("developer.ibm.com");
};
}

```

Because every request to your resource will create a new instance of `RSSAdapterResource`, it is important to reuse objects that may impact performance. In this example we made the Http client a `static` object and initialized it in a static `init()` method, which gets called by the `init()` of `RSSAdapterApplication` as described above.

Procedure resource

```

@GET
@Produces("application/json")
public void get(@Context HttpServletResponse response, @QueryParam("tag") String tag) throws ClientProtocolException, IOException, IllegalStateException, SAXException {
    if(tag!=null &&& !tag.isEmpty()){
        execute(new HttpGet("/mobilefirstplatform/tag/"+ tag +"/feed"), response);
    } else{
        execute(new HttpGet("/mobilefirstplatform/feed"), response);
    }
}

```

Our adapter exposes just one resource URL which allows to retrieve the RSS feed from the backend service.

- `@GET` means that this procedure only responds to `HTTP GET` requests.
- `@Produces("application/json")` specifies the Content Type of the response to send back. We chose to send the response as a `JSON` object to make it easier on the client-side.
- `@Context HttpServletResponse response` will be used to write to the response output stream. This enables us more granularity than returning a simple string.
- `@QueryParam("tag") String tag` enables the procedure to receive a parameter. The choice of `QueryParam` means the parameter is to be passed in the query (`/RSSAdapter/?tag=MobileFirst_Platform`). Other options include `@PathParam`, `@HeaderParam`, `@CookieParam`, `@FormParam`, etc.
- `throws ClientProtocolException, ...` means we are forwarding any exception back to the client. The client code is responsible for handling potential exceptions which will be received as `HTTP 500` errors. Another solution (more likely in production code) is to handle exceptions in your server Java code and decide what to send to the client based on the exact error.
- `execute(new HttpGet("/mobilefirstplatform/feed"), response)`. The actual HTTP request to the backend service is handled by another method defined later.

Depending if you pass a `tag` parameter, `execute` will retrieve a different build a different path and retrieve a different RSS file.

execute()

```
public void execute(HttpUriRequest req, HttpServletResponse resultResponse) throws ClientProtocolException, IOException,
IllegalStateException, SAXException {
    HttpResponse RSSResponse = client.execute(host, req);
    ServletOutputStream os = resultResponse.getOutputStream();

    if (RSSResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
        resultResponse.addHeader("Content-Type", "application/json");
        String json = XML.toJson(RSSResponse.getEntity().getContent());
        os.write(json.getBytes(Charset.forName("UTF-8")));</p>
    } else {
        resultResponse.setStatus(RSSResponse.getStatusLine().getStatusCode());
        RSSResponse.getEntity().getContent().close();
        os.write(RSSResponse.getStatusLine().getReasonPhrase().getBytes());
    }
    os.flush();
    os.close();
}
```

- `HttpResponse RSSResponse = client.execute(host, req)` . We use our static HTTP client to execute the HTTP request and store the response.
- `ServletOutputStream os = resultResponse.getOutputStream()` . This is the output stream to write a response to the client.
- `resultResponse.addHeader("Content-Type", "application/json")` . As mentioned before, we chose to send the response as JSON.
- `String json = XML.toJson(RSSResponse.getEntity().getContent())` . We used `org.apache.wink.json4j.utils.XML` to convert the XML RSS to a JSON string.
- `os.write(json.getBytes(Charset.forName("UTF-8")))` the resulting JSON string is written to the output stream.

The output stream is then `flushed` and `closed`.

If `RSSResponse` is not `200 OK`, we write the status code and reason in the response instead.

Results

Use the testing techniques described in Java Adapter (`./#testing`) to test your work.

The adapter should return the RSS feed converted to JSON.

```
{
  "rss": {
    "channel": {
      "description": "Develop, test, manage, and secure your mobile web, native and hybrid apps",
      "generator": "http://wordpress.org/?v=4.2.4",
      "item": [
        {
          "category": [
            "Mobile",
            "android",
            "Mobile Quality Assurance",
```

```

"mobile_development",
"mobilefirst",
"xamarin"
],
"commentRss": "https://developer.ibm.com/mobilefirstplatform/2015/09/01/integrating-mqa-into-xamarin-android-app/feed/",
"comments": [
  "https://developer.ibm.com/mobilefirstplatform/2015/09/01/integrating-mqa-into-xamarin-android-app/#comments",
  "0"
],
"creator": "Vidyasagar MSC",
"description": "<p>The post <a rel='nofollow' href='https://developer.ibm.com/mobilefirstplatform/2015/09/01/integrating-mqa-into-xamarin-android-app/'>Integrating MQA into Xamarin.Android app</a> appeared first on <a rel='nofollow' href='https://developer.ibm.com/mobilefirstplatform/'>IBM MobileFirst Platform</a>.</p>",
"encoded": "<p>It all startedÂ when I received an email seeking help on using MQA or to be more precise integrating MQA into Xamarin based android app. Before jumping into addressing the problem, let&#8217;s define MQA.</p>\n<h4>What is MQA?</h4>\n<p>MQA stands for &#8220;Mobile Quality Assurance&#8221; and is part of the IBM MobileFirst Platform.</p>\n<blockquote><p><em><span style='line-height: 1.5'>IBM MQA provides line of business professionals and development teams with insightful and streamlined quality feedback and metrics from both pre-production and production, enabling them to prioritize and take action to support a dynamic mobile app strategy.</span></em></p></blockquote>\n<p>The Features of MQA are</p>\n<div style='width: 1058px' class='wp-caption aligncenter'><a href='http://vidyasagarmsc.com/wp-content/uploads/2015/09/MQA1.png'><img class='size-full wp-image-65' src='http://vidyasagarmsc.com/wp-content/uploads/2015/09/MQA1.png' alt='Features of Mobile Quality Assurance.' width='1048' height='350' /></a><p class='wp-caption-text'>Features of Mobile Quality Assurance.</p></div>\n<p><em><strong>Note</strong></em>: To understand more about MQA, visitÂ <a href='http://www-03.ibm.com/software/products/en/ibm-mobilefirst-platform-quality-assurance'>IBM Mobile Quality Assurance</a></p>\n<p>So, by now we should be good with the first part of our blog title that is MQA. So, the next question is</p>\n<h4>What is Xamarin.Android?</h4>\n<p>Xamarin is a platform to create nativeÂ iOS, Android, Mac and Windows apps in C#.Â Xamarin.Android allows us to create native Android applications using the same UI controls we would in Java, except with the flexibility and elegance of a modern language (C#).</p>\n<p>As we are good with the definitions, let&#8217;s address the problem.</p>\n<p><strong>What&#8217;s the problem in integrating MQA into Xamarin Android app?</strong></p>\n<p>At the time of this blog post, the available MQA SDKs are iOS native SDK, Android native SDK and Javascript Â SDK.</p>\n<p>So, we have to find a workaround to address this use-case. The initial step is to download the Android MQA SDK and see what&#8217;s provided. you can download it from <a href='http://www-01.ibm.com/support/knowledgecenter/#!SSJML5_6.0.0/com.ibm.mqa.uau.saas.doc/topics/c_AndroidSDKsForDownload.html'>here</a>. Once successfully downloaded and unzipped, we should see a jar file namely <strong><em>MQA-Android-library-&#8217;s version number&#8217;s.jar</em>Â </strong>under lib folder<strong></strong></p>\n<div style='width: 634px' class='wp-caption aligncenter'><a href='http://vidyasagarmsc.com/wp-content/uploads/2015/09/MQA2.png'><img class='size-full wp-image-70' src='http://vidyasagarmsc.com/wp-content/uploads/2015/09/MQA2.png' alt='MQA Android SDK ' width='624' height='440' /></a><p class='wp-caption-text'>MQA Android SDK</p></div>\n<p>As Xamarin is C# based, What can we do with this jar file?</p>\n<p>We haveÂ <strong>Xamarin bindings</strong> to our rescue, which helps using in consuming .JARs from C#.</p>\n<p><strong><em>Note</em>:</strong> Steps to consume MQA Android JAR in a Xamarin.Android app is mentionedÂ <a href='https://developer.xamarin.com/guides/android/advanced_topics/java_integration_overview/binding_a_java_library_(.jar)'/>here</a></p>\n<div style='width: 257px' class='wp-caption aligncenter'><a href='http://vidyasagarmsc.com/wp-content/uploads/2015/09/MQA31.png'><img class='wp-image-72 size-full' src='http://vidyasagarmsc.com/wp-content/uploads/2015/09/MQA31.png' alt=' ' width='247' height='303' /></a><p class='wp-caption-text'>Xamarin binding project with MQA Android .JAR file</p></div>\n<p>The files of our interest here are <strong>MQA-Android-library-2.7.4.jar</strong> (Version number may vary) and <strong>Metadata.xml.</strong></p>\n<ul>\n<li>MQA-Android-library-2.7.4.jar file will have all the MQA related classes and methods required for us to start an Android MQA session </li>

```

We have all the metadata related classes and methods required for us to start an Android MQA session.

```
<?xml version='1.0' encoding='utf-8'>
<!-- Metadata.xml -->
<!-- Allows changes to be made to the final API, such as changing the namespace of the generated binding. -->
Based on the errors thrown while building the project, Metadata.xml in my case looks like this
<pre><?xml version='1.0' encoding='utf-8'>
<!-- Metadata.xml -->
<!-- This sample removes the class: android.support.v4.content.AsyncTaskLoader.LoadTask:
&lt;remove-node path=&quot;api/package[@name='android.support.v4.content']/class[@name='AsyncTaskLoader.LoadTask']&quot;/&gt;
This sample removes the method: android.support.v4.content.CursorLoader.loadInBackground:
&lt;remove-node path=&quot;api/package[@name='android.support.v4.content']/class[@name='CursorLoader']/method[@name='loadInBackground']&quot;/&gt;
--&gt;
&lt;remove-node path=&quot;api/package[@name='ext.com.google.inject.spi']/class[@name='InjectionPoint.Factory.1']&quot;/&gt;
&lt;remove-node path=&quot;api/package[@name='ext.com.google.inject.spi']/class[@name='InjectionPoint.Factory.2']&quot;/&gt;
&lt;remove-node path=&quot;api/package[@name='com.applause.android.log']/interface[@name='LoggerInterface']&quot;/&gt;
&lt;remove-node path=&quot;api/package[@name='ext.com.google.inject.internal']&quot;/&gt;
&lt;remove-node path=&quot;api/package[@name='ext.com.google.inject.matcher']&quot;/&gt;
&lt;remove-node path=&quot;api/package[@name='com.applause.android.util']/class[@name='AbstractRequest']&quot;/&gt;
&lt;remove-node path=&quot;api/package[@name='ext.com.google.inject.spi']/class[@name='Elements.RecordingBinder']/method[@name='bind' and count(parameter)=1 and parameter[1][@type='ext.com.google.inject.Key']]&quot;/&gt;
&lt;attr path=&quot;api/package[@name='com.applause.android.messages']/class[@name='Message']/field[@name='message']; name=managedName&quot;/&gt;
&lt;Message1&gt;
&lt;attr path=&quot;api/package[@name='com.applause.android.log']; name=managedName&quot;/&gt;
log&gt;
&lt;attr&gt;
&lt;metadata&gt;

```

Once all the errors are fixed and your binding project builds successfully, add a new Xamarin Android project (if you haven't added yet). Now, add MQA binding project reference in our Xamarin android app.

Note: Both your binding project and Xamarin.Android project should be of same target framework. You can verify this by right clicking on your project -> Options -> General.

Xamarin Android project with added reference to MQA

Now, let's start MQA Android session in our Count.Android app. Before doing this, we should create a MQA service on IBM Bluemix. You can follow the instructions mentioned at <https://www.ng.bluemix.net/docs/#services/MobileQualityAssurance/index.html#MobileQualityAssurance> Getting started with Mobile Quality Assurance - Bluemix or watch this video.

Starting a session with the Android SDK entails three steps. First, build a configuration to define how it works with your app. Second, start the session itself. Third, add tracking to your activities. Open MainActivity.cs file (Android Project) and paste the code provided below

```
<pre>using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace Count.Android
{
    [Activity(Label = "Count.Android", MainLauncher = true, Icon = "@drawable/icon")]
    public class MainActivity : Activity
    {
        int count = 1;

        void Use your own generated APP KEY
        const string APP_KEY = "1g59b7d884f9df5426162e5cb1f87a700648bce4fg0g1g379e0d3a";
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            //MQA Android session configuration
            Configuration configuration = new Configuration.Builder(this).WithAPIKey(APP_KEY).Provides the quality assurance application APP_KEY.WithMode(MQA.Mode.Qa).Selects the quality assurance application mode.WithReportOnShakeEnabled(true).Enables shake reporting.
            WithDefaultLog(&quot;com.ng.default_log"@email.com&quot;).Sets a default user agent.
        }
    }
}
```


m : which used to run JAX-RS application on Bluemix\nAdvanced Mobile Access service : which gives mobile application security and monitoring functionality\nPush Service for iOS 8 : which provides the capability to use iOS Push features\n\nLiberty Runtime </h3>\n\nLiberty contains two projects with JAX-RS service (i.e Custom-oauth-java for Custom Authentication and LocalstoreAdapter for storing items). The service include the protected resource and the custom identity provider code. The liberty server is configured with TAI.\nTrust Association Interface (TAI) is a service provider API that enables the integration of third-party security services with a Liberty profile server. For more info on TAI : click here\nThe custom identity provider authenticates a user by sending challenges to the client. However, custom identity providers do not communicate directly with clients. They send challenges and receive responses to the challenges by means of the Advanced Mobile Access service. When a custom identity provider successfully authenticates the user, it provides the user identity information to Advanced Mobile Access. For more information on custom authentication refer bluemix documentation : click here\n<p>The custom identity provider code is defined by two http API:</p>\n

```
\n
```

[illegible]

es Facebook, Google, or a custom identity provider to authenticate access to protected resources. Add Custom identity provider feature as it can be migrated to MFPF and specify the corresponding jax-rs custom authentication application url and realm name.



Custom Auth AMA

Add the following code inside didFinishLaunchingWithOptions function in AppDelegate of client application which will register the realm and initialize connection with Bluemix Application.

```
IMFClient.sharedInstance().registerAuthenticationDelegate(customAuthDelegate, forRealm: &quot;customAuthRealm_3&quot;)\nIMFClient.sharedInstance().initializeWithBackendRoute(&quot;https://parkstore.mybluemix.net&quot;, backendGUID: &quot;5e3ad88d-dd48-469d-b46f-2c4ad66b5345&quot;);
```

The following is the sample code to invoke the Rest url’s in client application.

```
var request: IMFResourceRequest = IMFResourceRequest(path: &quot;https://parkstore.mybluemix.net/LocalstoreAdapter/apps/5e3ad88d-dd48-469d-b46f-2c4ad66b5345/localstore/getAllItems&quot;, method: &quot;GET&quot;)\n    request.sendWithCompletionHandler { (wResponse:IMFResponse!, err:NSError!) -&gt; Void in\n
```

Push Service for iOS 8

Bind the application with Push Service for iOS 8



Push AMA

Configure Apple Push Notification service (APNs) which requires Apple Developer Account and Generate pl2 certificates. Documentation link : <https://www.ng.bluemix.net/docs/services/mobilepush/index.html#certificates> click here



Push Service

Add the following code inside didFinishLaunchingWithOptions function in AppDelegate of client application which will register notifications in client app.

```
let notificationTypes: UIUserNotificationType = UIUserNotificationType.Badge | UIUserNotificationType.Alert | UIUserNotificationType.Sound\n    let notificationSettings: UIUserNotificationSettings = UIUserNotificationSettings(forTypes: notificationTypes, categories: nil)\n    application.registerUserNotificationSettings(notificationSettings)\n    application.registerForRemoteNotifications()
```

Add the following code inside didRegisterForRemoteNotificationsWithDeviceToken function in AppDelegate of client application which will register pushclient and subscribe to tag in client app.

```
IMFPushClient.sharedInstance().registerDeviceToken(deviceToken, completionHandler: { (response, error) -&gt; Void in\n    if error != nil {\n        println(&quot;Error during device registration \\(error.description)&quot;)\n    } else {\n        println(&quot;Response during device registration json: \\(response.responseJson.description)&quot;)\n        var tags = [&quot;parkstore&quot;]\n        IMFPushClient.sharedInstance().subscribeToTags(tags, completionHandler: { (response:IMFResponse!, err:NSError!) -&gt; Void in\n            if err != nil {\n                println(&quot;There was an error while subscribing to tag&quot;)\n            } else {\n                println(&quot;Successfully subscribed to tag parkstore&quot;)\n            }\n        })\n    }\n})
```

Add the following function inside AppDelegate which triggers when push notification arrived in client app.

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject : AnyObject]) {\n    println(&quot;Got remote Notification. Data : \\(userInfo.description)&quot;)\n    let info = userInfo as NSDictionary\n    let data = info objectForKey(&quot;aps&quot;)?.objectForKey(&quot;alert&quot;) as! NSDictionary\n    let userData = data objectForKey(&quot;body&quot;) as! String\n    let alertView = UIAlertView(title: &quot;WishList&quot;, message: &quot;\\(userData)&quot;, delegate: nil, cancelButtonTitle: &quot;OK&quot;)\n    alertView.show()\n}
```

Existing Bluemix Client Application

Add the following Code snippets to the existing Bluemix Client Application and name the application with same name which you have registered in Advance Mobile Access Dashboard

Add the following code inside didFinishLaunchingWithOptions

ce Mobile Access Dashboard.<pre>Add the following code inside didFinishLaunchingWithOptions function in AppDelegate of client application which will register the realm and initialize connection with Bluemix Application.</pre>

```

IMFClient.sharedInstance().registerAuthenticationDelegate(customAuthDelegate, forRealm: &quot;customAuthRealm_3&quot;)
IMFClient.sharedInstance().initializeWithBackendRoute(&quot;https://parkstore.mybluemix.net&quot;, backendGUID: &quot;5e3ad88d-dd48-469d-b46f-2c4ad66b5345&quot;)

```

The following is the sample code to invoke the Rest url in client application.

```

var request: IMFResourceRequest = IMFResourceRequest(path: &quot;https://parkstore.mybluemix.net/LocalstoreAdapter/apps/5e3ad88d-dd48-469d-b46f-2c4ad66b5345/localstore/getAllItems&quot;, method: &quot;GET&quot;)
request.sendWithCompletionHandler { (wResponse:IMFResponse!, err:NSError!) -&gt; Void in

```

Add the following code inside didFinishLaunchingWithOptions function in AppDelegate of client application which will register notifications in client app.

```

let notificationTypes: UIUserNotificationType = UIUserNotificationType.Badge | UIUserNotificationType.Alert | UIUserNotificationType.Sound
let notificationSettings: UIUserNotificationSettings = UIUserNotificationSettings(forTypes: notificationTypes, categories: nil)
application.registerUserNotificationSettings(notificationSettings)
application.registerForRemoteNotifications()

```

Add the following code inside didRegisterForRemoteNotificationsWithDeviceToken function in AppDelegate of client application which will register pushclient and subscribe to tag in client app.

```

IMFPushClient.sharedInstance().registerDeviceToken(deviceToken, completionHandler: { (response, error) -&gt; Void in
    if error != nil {
        println(&quot;Error during device registration \((error.description)&quot;)
    } else {
        println(&quot;Response during device registration json: \((response.responseJson.description)&quot;)
        var tags = [&quot;parkstore&quot;]
        IMFPushClient.sharedInstance().subscribeToTags(tags, completionHandler: { (response:IMFResponse!, err:NSError!) -&gt; Void in
            if err != nil {
                println(&quot;There was an error while subscribing to tag&quot;)
            } else {
                println(&quot;Successfully subscribe to tag parkstore&quot;)
            }
        })
    }
}

```

Add the following function inside AppDelegate which triggers when push notification arrived in client app.

```

func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject : AnyObject]) {
    println(&quot;Got remote Notification. Data : \((userInfo.description)&quot;)
    let info = userInfo as! NSDictionary
    let data = info objectForKey(&quot;aps&quot;)?.objectForKey(&quot;alert&quot;) as! NSDictionary
    let userData = data objectForKey(&quot;body&quot;) as! String
    let alertView = UIAlertView(title: &quot;WishList&quot;, message: &quot;\((userData)&quot;, delegate: nil, cancelButtonTitle: &quot;OK&quot;)
    alertView.show()
}

```

The following are the screenshots of client application.









Migration to On-Prem

Migration of Client Application

Migration of Client Application includes following two steps

- Configuring Cocoapods
- Client App Migration

Configuring Cocoapods

If CocoaPods has not been installed on a specific computer:

Follow the “Getting Started” guide for CocoaPods installation

n: <http://guides.cocoapods.org/using/getting-started.html> Open “Terminal” at the installation location and run the “pod init” command

The following steps assume that the client application is working with CocoaPods. If not, follow this “Using CocoaPods” documentation : <http://guides.cocoapods.org/using/using-cocoapods.html> click here

In both cases, the instructions below explain how to edit the “Podfile” file.

Open the “Podfile” file located in the root of your XCode project in a favourite text editor.

Comment out or remove the existing content.

Add the following lines:

```
source 'https://github.com/rleigh.ibm.com/imflocalsdks/imf-client-sdk-specs.git'
pod 'IMFCompatibility'
```

Open “Terminal” at the location of “Podfile”.

Verify that the XCode project is closed.

Run the “pod install” command.

Open the [MyProject].xcworkspace file in XCode. This file is located side by side with [MyProject].xcodeproj.

An usual CocoaPods-based project is managed as a workspace containing the application (the executable) and the library (all project dependencies brought by the CocoaPods manager).

In Xcode's Build Settings, search for “Other Linker Flags” and insert \${inherited} (if -ObjC is defined in this field, you can just delete it, since it is configured in the CocoaPod project).

Client App Migration

Search for bluemix dependency imports like

```
#import <IMFCore/IMFCore.h>
#import <IMFPush/IMFPush.h>
```

Replace the above imports with

```
#import <IMFCompatibility/IMFCompatibility.h>
```

Look for a call to the “initWithBackendRoute” method and replace the route URL with your on-premise server URL. For example:

```
IMFClient.sharedInstance().initWithBackendRoute("https://parkstore.mybluemix.net", backendGUID: "5e3ad88d-dd48-469d-b46f-2c4ad66b5345");
```

should be replaced with your on-premise MFP server URL

```
IMFClient.sharedInstance().initWithBackendRoute("http://localhost:10080/ParkStoreMFP", backendGUID: "5e3ad88d-dd48-469d-b46f-2c4ad66b5345");
```

Note, that backendGUID parameter is ignored and can be empty. Look for all instantiations of IMFResourceRequest class and update it

Look for all instantiations of IMFResourceRequest class and update the request URL with absolute or relative path to the resource. For example:

```
var request: IMFResourceRequest = IMFResourceRequest(path: "https://parkstore.mybluemix.net/LocalstoreAdapter/apps/5e3ad88d-dd48-469d-b46f-2c4ad66b5345/Localstore/getAllItems", method: "GET");
```

should be replaced with

```
var request: IMFResourceRequest = IMFResourceRequest(path: "http://localhost:10080/ParkStoreMFP/adapters/LocalstoreAdapter/Localstore/getAllItems", method: "GET");
```

Add the following code inside didRegisterForRemoteNotificationsWithDeviceToken function in AppDelegate of Client application.

```
WLPush.sharedInstance().tokenFromClient = deviceToken.description
```

All on-premise applications require the “worklight.plist” file to be present in the application resources. In the <code>IBMMobileFirstPlatformFoundationNativeSDK</code> pod we supply a file named sample.worklight.plist.

Locate the “sample.worklight.plist” file in the "IBMMobileFirstPlatformFoundationNativeSDK" pod.

Copy this file to the parent (application) project and rename it to “worklight.plist”.

Edit the “worklight.plist” file by setting the “application id” key to the name of your application deployed to the on-premise MFPF server

Migration of JAX-RS Application to JAVA Adapter

To migrate JAX-RS application to on-prem (MobileFirst Foundation) server we need to do the following steps for server:

Create MobileFirst Project – Create native API app for iOS

<https://developer.ibm.com/mobilefirstplatform/wp-content/uploads/sites/32/2015/07/Screen-Shot-2015-07-12-at-6.50.04-pm.png>



<https://developer.ibm.com/mobilefirstplatform/wp-content/uploads/sites/32/2015/07/Screen-Shot-2015-07-12-at-6.51.13-pm.png>



14818" V><Va><Vp>\n<p><Va><Vli>\nAdd two adapters for Custom Authentication and Localstore and migrate the JAX-RS code as shown in the following example.\n\n<p>Copy the JAX-RS BlueMix code and paste it in the newly created Localstore Java adapter JAX-RS file.</p>\n<p>Add and remove the following changes in your adapter code.</p>\n\nremove <code>V{tenantId}</code>\nremove the <code>@PathParam -> PathParam("tenantId") String deviceId</code> and <code>@PathParam("realmName") String realmName</code>\nAdd scope to the all http api resource <code>@OAuthSecurity (scope="customAuthRealm_3")</code>\n\n<p>The code looks like the following</p>\n<pre class="brush: plain; title: ; notranslate">\n\t@GET\n\t@OAuthSecurity (scope="customAuthRealm_3"")\n\t@Path("VgetAllItems")\n\tpublic String getAllItems() throws MissingConfigurationException{\n\t\tinit();\n\t\tJSONArray jsonArray = new JSONArray();\n\t\tfor(Object key : props.keySet()){ \n\t\t\tjsonArray.add(parser.parse(props.getProperty((String) key)).getAsJsonObject());\n\t\t}\n\t\treturn jsonArray.toString();\n\t}\n\t@PUT\n\t@OAuthSecurity (scope="customAuthRealm_3"")\n\t@Path("VaddItem")\n\tpublic void addItem(String itemJson) \n\t\tthrows MissingConfigurationException, URISyntaxException, IOException{\n\t\ttry{\n\t\t\tinit();\n\t\t\tint newKey = props.keySet().size()+1;\n\t\t\tprops.put(String.valueOf(newKey), itemJson);\n\t\t\tURL url = this.getClass().getClassLoader().getResource("data.properties");\n\t\t\tFile file = new File(url.toURI().getPath());\n\t\t\tFileOutputStream foStream = new FileOutputStream(file);\n\t\t\tprops.store(foStream, "saving new item");\n\t\t\tfoStream.close();\n\t\t} catch(IOException ioe){\n\t\t\tioe.printStackTrace();\n\t\t}\n\t\t@POST\n\t\t@OAuthSecurity (scope="customAuthRealm_3"")\n\t\t@Path("VaddAllItems")\n\t\tpublic String addAllItems(String itemsJson) \n\t\t\tthrows MissingConfigurationException, URISyntaxException, IOException{\n\t\t\ttry{\n\t\t\t\tinit();\n\t\t\t\tclearAllData();\n\t\t\t\tJSONArray jsonArr = parser.parse(itemsJson).getAsJSONArray();\n\t\t\t\tfor(int i=0;i<jsonArr.size();i++){ \n\t\t\t\t\tprops.put(String.valueOf(i+1), jsonArr.get(i).toString());\n\t\t\t\t}\n\t\t\t\tURL url = this.getClass().getClassLoader().getResource("data.properties");\n\t\t\t\tFile file = new File(url.toURI().getPath());\n\t\t\t\tFileOutputStream foStream = new FileOutputStream(file);\n\t\t\t\tprops.store(foStream, "saving new item");\n\t\t\t\tfoStream.close();\n\t\t\t\treturn "";\n\t\t\t} catch(IOException ioe){\n\t\t\t\tioe.printStackTrace();\n\t\t\t\treturn "";\n\t\t\t}\n\t\t@DELETE\n\t\t@OAuthSecurity(enabled=false)\n\t\t@Path("VclearAll")\n\t\tpublic String clearAllData() \n\t\t\tthrows MissingConfigurationException, URISyntaxException, IOException{\n\t\t\tinit();\n\t\t\tprops.clear();\n\t\t\tSystem.out.println("Size : "+props.size());\n\t\t\tURL url = this.getClass().getClassLoader().getResource("data.properties");\n\t\t\tFile file = new File(url.toURI().getPath());\n\t\t\tFileOutputStream foStream = new FileOutputStream(file);\n\t\t\tprops.store(foStream, "clearing all data");\n\t\t\tfoStream.close();\n\t\t\treturn "cleared";\n\t\t}\n\t</pre>\n<h3 id="configoauth">Configuring Custom-OAuth</h3>\n\nAdd realm with same name you had on BlueMix and login module to the authenticationC onfig.xml.\n<pre class="brush: xml; title: ; notranslate">\n<realm name="customAuthRealm_3"" loginModule="customAuthLoginModule_3"">\n<class Name>com.worklight.core.auth.ext.CustomIdentityAuthenticator</class Name>\n<parameter name="providerUrl" value="http://localhost:10080/ParkStoreMFP/adapters/Customauth">\n</realm>\n<loginModule name="customAuthLoginModule_3"" expirationInSeconds="3600">\n<class Name>com.worklight.core.auth.ext.CustomIdentityLoginModule</class Name>\n</loginModule>\n</pre>\n\nAdd Custom-oauth Realm in userIdentityRealms in Application Descriptor file of iOS Native API\n<pre class="brush: xml; title: ; notranslate">\n<userIdentityRealms>\n <customAuthRealm_3>\n <userIdentityRealms>\n </customAuthRealm_3>\n </customAuthRealm_3>\n</pre>\n\n\n<h3 id="configurepush">Configuring Push Capability</h3>\n\nAdd a p12 certificate which is generated from Apple Developer Account under iOS Native API Folder\n<p><Va>\n\nAdd Push configuration in Application Descriptor file of iOS Native API and include the pas

