Resource Request from Native iOS Swift Applications

Overview

MobileFirst applications can access resources using the `WLResourceRequest` REST API.
The REST API works with all adapters and external resources.

Prerequisite: Ensure you have added the MobileFirst Platform SDK (../../adding-the-mfpf-sdk/adding-the-mfpf-sdk-to-ios-applications) to your Native iOS project.

WLResourceRequest

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

Create a `WLResourceRequest` object and specify the path to the resource and the HTTP method.
Available methods are: `WLHttpMethodGet`, `WLHttpMethodPost`, `WLHttpMethodPut` and `WLHttpMethodDelete`.

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

- For **JavaScript adapters**, use `/adapters/{AdapterName}/{procedureName}`
- For **Java adapters**, use `/adapters/{AdapterName}/{path}`. The `path` depends on how you defined your `@Path` annotations in your Java code. This would also include any `@PathParam` you used.
- To access resources outside of the project, use the full URL as per the requirements of the external server.

Sending the request

Request the resource by using the `sendWithCompletionHandler` method.
Supply a completion handler to handle the retrieved data:

```
request.sendWithCompletionHandler { (WLResponse response, NSError error) -> Void in
    var resultText = ""
    if(error != nil){
        resultText = "Failed to call the resource"
        resultText += error.description
    }
    else if(response != nil){
        resultText = "Successfully called the resource"
        resultText += response.responseText
    }
    self.updateView(resultText)
}
```

Use the `response` and `error` objects to get the data that is retrieved from the adapter.

The `response` object contains the response data and you can use its methods and properties to retrieve the required information. Commonly used properties are `responseText -> String`, `responseJSON -> Dictionary` (if the response is in JSON) and `status -> Int` (the HTTP status of the response).

Alternatively, you can use `sendWithDelegate` and provide a delegate that conforms to both the `NSURLConnectionDataDelegate` and `NSURLConnectionDelegate` protocols. This will allow you to handle the response with more granularity, such as handling binary responses.

Parameters

Before sending your request, you may want to add parameters as needed.

Path parameters

As explained above, **path** parameters (`/path/value1/value2`) are set during the creation of the `WLResourceRequest` object.

Query parameters

To send **query** parameters (`/path?param1=value1...`) use the `setQueryParameter` method for each parameter:

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

Form parameters

To send **form** parameters in the body, use `sendWithFormParameters` instead of the simple `sendWithCompletionHandler`:

```
//@FormParam("height")
let formParams = ["height":"175"]

//Sending the request with Form parameters
request.sendWithFormParameters(formParams) { (response, error) -> Void in
    if(error == nil){
        NSLog(response.responseText)
    }
    else{
        NSLog(error.description)
    }
}
```

Header parameters

To send a parameter as an HTTP header use the `setHeaderValue` API:

```
//@HeaderParam("Date")
request.setHeaderValue("2015-06-06", forName: "Date")
```

Other custom body parameters

- `sendWithBody` allows you to set an arbitrary String in the body.
- `sendWithJSON` allows you to set an arbitrary dictionary in the body.
- `sendWithData` allows you to set an arbitrary `NSData` in the body.

Javascript Adapters

JavaScript adapters use ordered nameless parameters. To pass parameters to a Javascript adapter, set an array of parameters with the name `params`:

```
request.setQueryParameterValue("['param1', 'param2']", forName: "params")
```

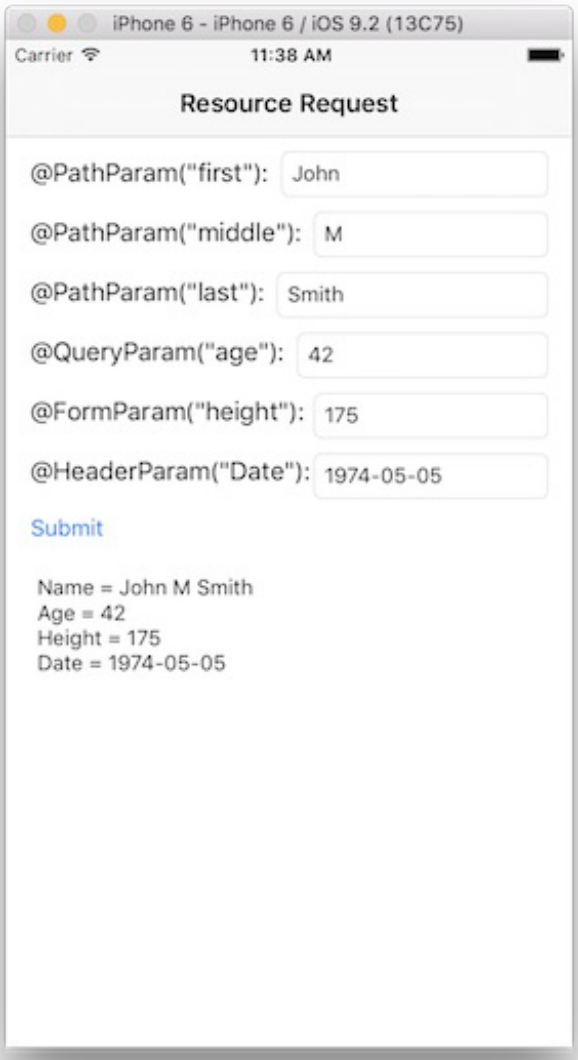
For more information

> For more information about WLResourceRequest, refer to the user documentation.

Sample application

The ResourceRequestSwift project contains a native iOS Swift application that makes a resource request using a Java adapter.
The adapter Maven project contains the Java adapter to be used during the resource request call.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/ResourceRequestSwift/tree/release80) the Native project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/Adapters/tree/release80) the adapter Maven project.



Sample usage

- Make sure to update the **mfpclient.plist** file in the Xcode project with the server properties.
- The sample uses the `JavaAdapter` contained in the Adapters Maven project. Use either Maven or MobileFirst Developer CLI to build and deploy the adapter (../../creating-adapters/).