

Testing and Debugging Adapters

Overview

You can test Java and JavaScript adapters as well as debug Java code implemented for use in Java or JavaScript adapters via IDEs such as Eclipse, IntelliJ and alike.

This tutorial demonstrates how to test adapters using the MobileFirst Developer CLI and using Postman and also how to debug a Java adapter using the Eclipse IDE.

Jump to:

- Testing Adapters
 - Using Postman
 - Using Swagger
- Debugging Adapters
 - JavaScript adapters
 - Java adapters

Testing Adapters

MobileFirst adapters are available via a REST interface. This means that if you know the URL of a resource, you can use HTTP tools such as Postman to test requests and pass URL parameters, path parameters, body parameters or headers as you see fit.

The structure of the URL used to access the adapter resource is:

- In JavaScript adapters - `http://hostname-or-ip-address:port-number/mfp/api/adapters/{adapter-name}/{procedure-name}`
- In Java adapters - `http://hostname-or-ip-address:port-number/mfp/api/adapters/{adapter-name}/{path}`

Passing parameters

- When using Java adapters, parameters can be passed in the URL, body, form, etc, depending on how you configured your adapter.
- When using JavaScript adapters, parameters are passed as `params=["param1", "param2"]`. In other words, a JavaScript procedure receives only one parameter called `params` which needs to be an array of ordered, unnamed values. This parameter can either be in the URL (GET) or in the body (POST) using `Content-Type: application/x-www-form-urlencoded`.

Handling security

If your resource is protected by a scope, the request prompts you to provide a valid authorization header. Note that by default, MobileFirst uses a simple security scope even if you did not specify any. So unless you specifically disabled security, the endpoint is always protected.

To disable security in Java adapters you should attach the `OAuthSecurity` annotation to the method/class:

```
@OAuthSecurity(enabled=false)
```

To disable security in JavaScript adapters you should add the `secured` attribute to the procedure:

```
<procedure name="adapter-procedure-name" secured="false"/>
```

Alternatively, the development version of the MobileFirst Server includes a test token endpoint to bypass the security challenges.

Using Postman

Test Token

To receive a Test Token you should:

1. Use your HTTP client (Postman) to make an HTTP `POST` request to `http://<IP>:<PORT>/mfp/api/az/v1/token` with the following parameters using `Content-Type: application/x-www-form-urlencoded`:

```
grant_type : client_credentials
scope : **
```



2. Add an `authorization` header using `Basic authentication` with username "test" and password "test".



Using Swagger

The Swagger docs UI is a visual representation of an adapter's REST endpoints. Using Swagger, a developer can test the adapter endpoints before they are consumed by a client application.

To access Swagger:

1. Open the MobileFirst Operations Console and select an adapter from the adapters list.
2. Click on the **Resources** tab.
3. Click on the **View swagger Docs** button.
4. Click on the **Show/Hide** button.



SampleAdapter

Adapter resource reference

default

Show/Hide | List Operations | Expand Operations

GET	/users
GET	/users/helloUserQuery
PUT	/users/newUsers
POST	/users/sendNotification
POST	/users/{first}/{middle}/{last}
GET	/users/{username}

[BASE URL: /mfp/api/adapters/SampleAdapter]

Test Token

To add a Test Token to the request, so the security framework will skip any security challenges protecting your resource, click the **on/off switch** button on the right corner of an endpoint's operation.



You will be asked to select which scopes you want to grant to the Swagger UI (for testing purposes you can select all). If you are using the Swagger UI for the first time you may be required to log in with the Confidential Client ID ("test") and Secret ("test").

Learn more about Confidential Client in the user documentation.

Sending request

Expand the endpoint's operation, enter the required parameters (if needed) and click on the **Try it out!** button.

POST

/users/{first}/{middle}/{last}

ON

Response Class (Status 200)

Response Content Type */*

Parameters

Parameter	Value	Description	Parameter Type	Data Type
first	John		path	string
middle	M		path	string
last	Smith		path	string
age	42		query	integer
height	175		formData	string
Date	1974-05-05		header	string

Try it out!

Hide Response

Curl

curl -X POST --header "Content-Type: application/x-www-form-urlencoded" --header "Accept: */*" --header "Date: 1974-05-

Request URL

http://localhost:9080/mfp/api/adapters/SampleAdapter/users/John/M/Smith?age=42

Response Body

John M Smith
Age: 42
Height: 175
Date: null

Response Code

200

Response Headers

{
 "date": "Wed, 03 Feb 2016 07:58:31 GMT",
 "x-powered-by": "Servlet/3.1",
 "content-length": "43",
 "content-type": "application/octet-stream"
}

Debugging Adapters

JavaScript adapters

You can debug JavaScript code in JavaScript adapters by using the `WL.Logger` API.

Available logging levels, from least to most verbose, are: `WL.Logger.error`, `WL.Logger.warn`, `WL.Logger.info` and `WL.Logger.debug`.

The logs are then printed to the log file of the application server.

Be sure to set the server verbosity level accordingly, otherwise you will not see the logging in the log file.

Java adapters

Before an adapter's Java code can be debugged, Eclipse needs to be configured as follows:

1. **Maven integration** - Starting Eclipse Kepler (v4.3), Maven support is built-in in Eclipse. If your Eclipse instance does not support Maven, follow the m2e instructions (<http://www.eclipse.org/m2e/>) to add Maven support.
2. Once Maven is available in Eclipse, import the adapter Maven project:



3. Provide debugging parameters:

- Click **Run** → **Debug Configurations**.
- Double-click on **Remote Java application**.
- Provide a **Name** for this configuration.
- Set the **Port** value to "10777".
- Click **Browse** and select the Maven project.
- Click **Debug**.



4. Click on **Window** → **Show View** → **Debug** to enter *debug mode*. You can now debug the Java code normally as you would do in a standard Java application. You need to issue a request to the adapter to make the code run and hit any set breakpoints. This can be accomplished by following the instructions on how to call an adapter resource in the Testing adapters section ([../creating-adapters/#testing-adapters](#)).



For instructions how to use IntelliJ to debug Java adapters see the [Using IntelliJ to Develop MobileFirst Java Adapters](http://file:///home/travis/build/MFPSamples/DevCenter/_site/blog/2016/03/21/using-intellij-to-develop-adapters) Blog Post.