JavaScript HTTP Adapter

Overview

By using IBM MobileFirst Foundation HTTP adapters, you can send GET or POST HTTP requests and retrieve data from the response headers and body. HTTP adapters work with RESTful and SOAP-based services, and can read structured HTTP sources such as RSS feeds.

You can easily customize HTTP adapters with simple server-side JavaScript code. For example, you could set up server-side filtering if necessary. The retrieved data can be in XML, HTML, JSON, or plain text format.

The adapter is configured with XML to define the adapter properties and procedures.
Optionally, it is also possible to use XSL to filter received records and fields.

Prerequisite: Make sure to read the JavaScript Adapters (../) tutorial first.

The XML File

The XML file contains settings and metadata.
To edit the adapter XML file, you must:

- Set the protocol to HTTP or HTTPS.
- Set the HTTP domain to the domain part of HTTP URL.
- Set the TCP Port.

Declare the required procedures below the `connectivity` element:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<mfp:adapter name="JavaScriptHTTP"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>JavaScriptHTTP</displayName>
    <description>JavaScriptHTTP</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>https</protocol>
            <domain>mobilefirstplatform.ibmcloud.com</domain>
            <port>443</port>
            <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
            <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
            <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
        </connectionPolicy>
    </connectivity>

    <procedure name="getFeed"/>
    <procedure name="getFeedFiltered"/>
</mfp:adapter>
```

JavaScript implementation

A service URL is used for procedure invocations. Some parts of the URL are constant; for example, http://example.com/.
Other parts of the URL can be parameterized; that is, substituted at run time by parameter values that are provided to the MobileFirst procedure.

The following URL parts can be parameterized.

- Path elements
- Query string parameters
- Fragments

> See the topic about "The connectionPolicy element of the HTTP adapter" in the user documentation for advanced options for adapters, such as cookies, headers, and encoding.

To call an HTTP request, use the `MFP.Server.invokeHttp` method.
Provide an input parameter object, which must specify:

- The HTTP method: `GET`,`POST`, `PUT`, `DELETE`
- The returned content type: `XML`, `JSON`, `HTML`, or `plain`
- The service `path`
- The query parameters (optional)
- The request body (optional)
- The transformation type (optional)

```
function getFeed() {
  var input = {
      method : 'get',
      returnedContentType : 'xml',
      path : "feed.xml"
  };


  return MFP.Server.invokeHttp(input);
}
```

> See the topic about "MFP.Server.invokeHttp" in the user documentation for a complete list of options.

XSL transformation filtering

You can also apply XSL transformation to the received data, for example to filter the data.
To apply XSL transformation, create a **filtered.xsl** file next to the JavaScript implementation file.

You can then specify the transformation options in the input parameters of the procedure invocation. For example:

```
function getFeedFiltered() {

  var input = {
      method : 'get',
      returnedContentType : 'xml',
      path : "feed.xml",
      transformation : {
        type : 'xslFile',
        xslFile : 'filtered.xsl'
      }
  };

  return MFP.Server.invokeHttp(input);
}
```

> For more information on XSL transformation, refer to the user documentation.

Creating a SOAP-based service request

You can use the `MFP.Server.invokeHttp` API method to create a **SOAP** envelope.
Note: To call a SOAP-based service in a JavaScript HTTP adapter, you can encode the SOAP XML envelope within the request body using **E4X**.

```
var request =
      <soap:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body>
          <GetCitiesByCountry xmlns="http://www.webserviceX.NET">
            <CountryName>{countryName}</CountryName>
          </GetCitiesByCountry>
        </soap:Body>
      </soap:Envelope>;
```

The `MFP.Server.invokeHttp(options)` method is then used to call a request for a SOAP service.
The Options object must include the following properties:

- A `method` property: usually `POST`
- A `returnedContentType` property: usually `XML`
- A `path` property: a service path
- A `body` property: `content` (SOAP XML as a string) and `contentType`

```
var input = {
    method: 'post',
    returnedContentType: 'xml',
    path: '/globalweather.asmx',
    body: {
        content: request.toString(),
        contentType: 'text/xml; charset=utf-8'
    }
};

var result = MFP.Server.invokeHttp(input);
```

Invoking results of SOAP-based service

The result is wrapped into a `JSON` object:


```
{
    "statusCode" : 200,
    "errors" : [],
    "isSuccessful" : true,
    "statusReason" : "OK",
    "Envelope" : {
        "Body" : {
            "GetWeatherResponse" : {
                "xmlns" : "http://www.webserviceX.NET",
                "GetWeatherResult" : "<?xml version=\"1.0\" encoding=\"utf-16\"?>\n<CurrentWeather>\n  <L
ocation>Shanghai / Hongqiao, China (ZSSS) 31-10N 121-26E 3M</Location>\n  <Time>Mar 07, 2016 - 01
:30 AM EST / 2016.03.07 0630 UTC</Time>\n  <Wind> from the W (270 degrees) at 4 MPH (4 KT) (directi
on variable):0</Wind>\n  <Visibility> 4 mile(s):0</Visibility>\n  <Temperature> 69 F (21 C)</Temperature>\
n  <DewPoint> 53 F (12 C)</DewPoint>\n  <RelativeHumidity> 56%</RelativeHumidity>\n  <Pressure> 29.
94 in. Hg (1014 hPa)</Pressure>\n  <Status>Success</Status>\n</CurrentWeather>"
            }
        },
        "xsd" : "http://www.w3.org/2001/XMLSchema",
        "soap" : "http://schemas.xmlsoap.org/soap/envelope/",
        "xsi" : "http://www.w3.org/2001/XMLSchema-instance"
    },
    "responseHeaders" : {
        "X-AspNet-Version" : "4.0.30319",
        "Date" : "Mon, 07 Mar 2016 06:46:08 GMT",
        "Content-Length" : "1027",
        "Content-Type" : "text/xml; charset=utf-8",
        "Server" : "Microsoft-IIS/7.0",
        "X-Powered-By" : "ASP.NET",
        "Cache-Control" : "private, max-age=0",
        "X-RBT-Optimized-By" : "e8i-wx-sh4 (RiOS 8.6.2d-ibm1) SC"
    },
    "warnings" : [],
    "totalTime" : 654,
    "responseTime" : 651,
    "info" : []
}
```


Note the `Envelope` property, which is specific of SOAP-based requests.
The `Envelope` property contains the result content of the SOAP-based request.

To access the XML content:

- On client-side, jQuery can be used to wrap the result string, and follow the DOM nodes:

```
WL.Client.invokeProcedure({
    adapter : "JavaScriptSOAP",
    procedure : "getWeatherInfo",
    parameters : [ 'Washington', 'United States' ]
}, {
    onSuccess : function(resp) {
        var $result = $(resp.invocationResult.Envelope.Body.GetWeatherResponse.GetWeatherResult);
        var weatherInfo = {
            location: $result.find('Location').text(),
            time: $result.find('Time').text(),
            wind: $result.find('Wind').text(),
            temperature: $result.find('Temperature').text(),
        };
    }
});
```

- On server-side, create an XML object with the result string. The nodes can then be accessed as properties:

```
var xmlDoc = new XML(result.Envelope.Body.GetWeatherResponse.GetWeatherResult);
var weatherInfo = {
    Location: xmlDoc.Location.toString(),
    Time: xmlDoc.Time.toString(),
    Wind: xmlDoc.Wind.toString(),
    Temperature: xmlDoc.Temperature.toString()
};
```

Sample adapter

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

Sample usage

- Use either Maven, MobileFirst CLI or your IDE of choice to build and deploy the JavaScriptHTTP adapter (../../creating-adapters/).
- To test or debug an adapter, see the testing and debugging adapters (../../testing-and-debugging-adapters) tutorial.