Advanced Adapter Usage and Mashup

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/adapters/advanced-adapter-usage-mashup/index.md) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

Now that basic usage of different types of adapters has been covered, it is important to remember that adapters can be combined to make a procedure that uses different adapters to generate one processed result. You can combine several sources (different HTTP servers, SQL, etc).

In theory, from the client side, you could make several requests successively, one depending on the other. However, writing this logic on the server side could be faster and cleaner.

Jump to:

- JavaScript adapter API
- Java adapter API
- Data mashup example
- Sample application

JavaScript adapter API

Calling a JavaScript adapter procedure from a JavaScript adapter

When calling a JavaScript adapter procedure from another JavaScript adapter use the `WL.Server.invokeProcedure(invocationData)` API. This API enables to invoke a procedure on any of your JavaScript adapters. `WL.Server.invokeProcedure(invocationData)` returns the result object retrieved from the called procedure.

The `invocationData` function signature is:
`WL.Server.invokeProcedure({adapter: [Adapter Name], procedure: [Procedure Name], parameters: [Parameters seperated by a comma]})`

For example:

```
WL.Server.invokeProcedure({ adapter : "AcmeBank", procedure : " getTransactions", parameters : [account Id, fromDate, toDate], });
```

> Calling a Java adapter from a JavaScript adapter is not supported

Java adapter API

Before you can call another adapter - the AdaptersAPI must be assigned to a variable:

```
@Context
AdaptersAPI adaptersAPI;
```

Calling a Java adapter from a Java adapter

When calling an adapter procedure from a Java adapter use the `executeAdapterRequest` API. This call returns an `HttpResponse` object.

```
HttpUriRequest req = new HttpGet(getWeatherInfoProcedureURL);
org.apache.http.HttpResponse response = adaptersAPI.executeAdapterRequest(req);
JSONObject jsonObj = adaptersAPI.getResponseAsJSON(response)
```

Calling a JavaScript adapter procedure from a Java adapter

When calling a JavaScript adapter procedure from a Java adapter use both the `executeAdapterRequest` API and the `createJavascriptAdapterRequest` API that creates an `HttpUriRequest` to pass as a parameter to the `executeAdapterRequest` call.

```
HttpUriRequest req = adaptersAPI.createJavascriptAdapterRequest(AdapterName, ProcedureName, [para
meters]);
org.apache.http.HttpResponse response = adaptersAPI.executeAdapterRequest(req);
JSONObject jsonObj = adaptersAPI.getResponseAsJSON(response);
```

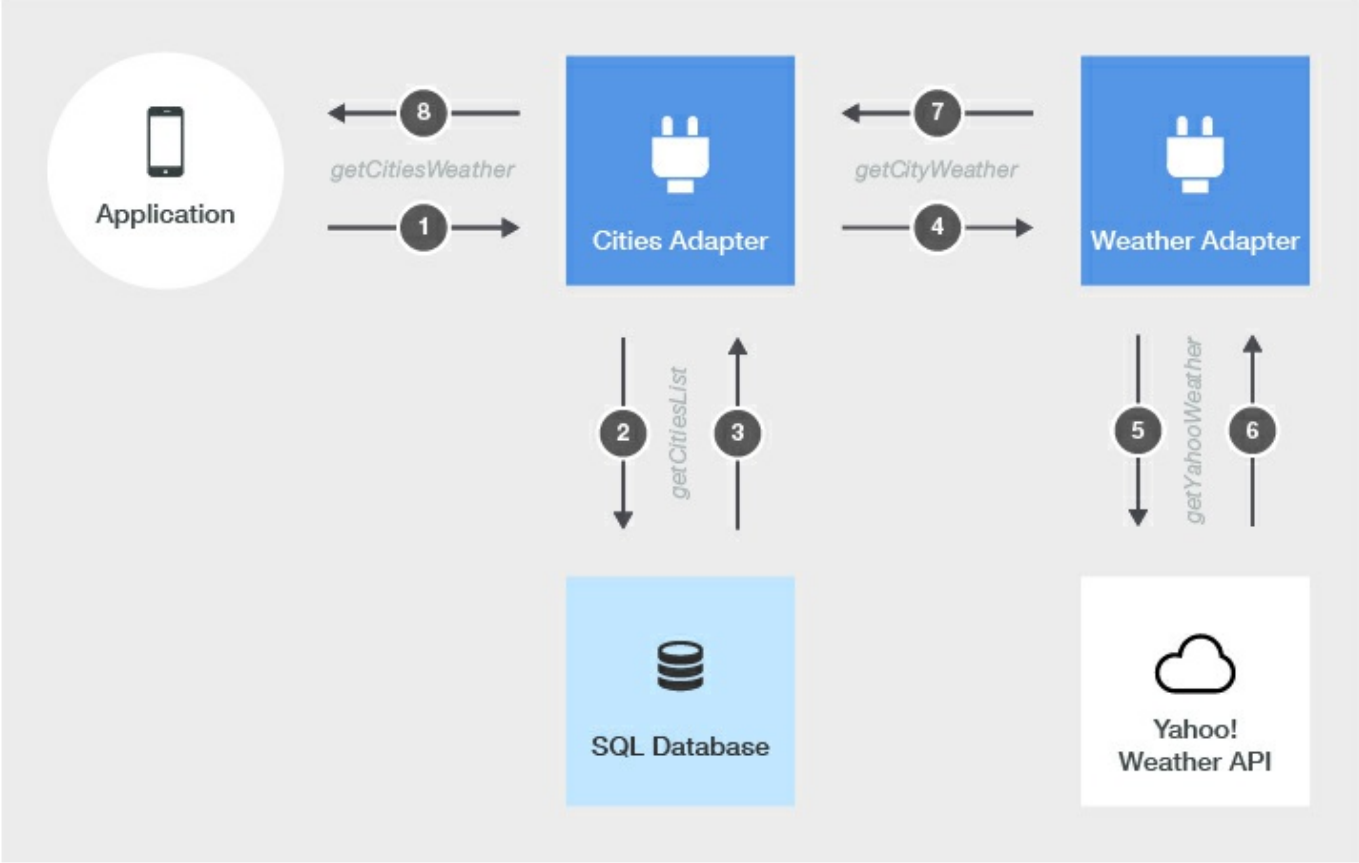
Data mashup example

The following example shows how to mash up data from 2 data sources, a database table and Yahoo! Weather Service, And to return the data stream to the application as a single object.

In this example we will use 2 adapters:

- Cities Adapter:
  - Extract a list of cities from a “weather” database table.
  - The result contains the list of several cities around the world, their Yahoo! Weather identifier and some description.
- Weather Adapter:
  - Connect to the Yahoo! Weather Service.
  - Extract an updated weather forecast for each of the cities that are retrieved via the Cities adapter.

Afterward, the mashed-up data is returned to the application for display.



The provided sample in this tutorial demonstrates the implementation of this scenario using 3 different mashup types.
In each one of them the names of the adapters are slightly different.
Here is a list of the mashup types and the corresponding adapter names:

Scenario	Cities Adapter name	Weather Adapter name
JavaScript adapter → **JavaScript** adapter	getCitiesListJS	getCityWeatherJS
Java adapter → **JavaScript** adapter	getCitiesListJavaToJS	getCityWeatherJS
Java adapter → **Java** adapter	getCitiesListJava	getCityWeatherJava

Mashup Sample Flow

1. Create a procedure / adapter call that create a request to Yahoo! Weather Service for each city and retrieves the corresponding data:

(getCitiesListJS adapter) XML:

```
<connectivity>
   <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>weather.yahooapis.com</domain>
      <port>80</port>
      ...
   </connectionPolicy>
</connectivity>
```

(getCitiesListJS adapter) JavaScript:

```
function getYahooWeather(woeid) {
    var input = {
        method : 'get',
        returnedContentType : 'xml',
        path : 'forecastrss',
        parameters : {
            'w' : woeid,
            'u' : 'c' //celcius
        }
    };

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

(getCityWeatherJava adapter)

```
@GET
@Produces("application/json")
public Response get(@QueryParam("cityId") String cityId) throws ClientProtocolException, IOException, Ill
egalStateException, SAXException {
    Response returnValue = execute(new HttpGet("/forecastrss?w="+ cityId +"&u=c"));
    return returnValue;
}

private Response execute(HttpUriRequest req) throws ClientProtocolException, IOException, IllegalStateE
xception, SAXException {
    HttpResponse RSSResponse = client.execute(host, req);
    if (RSSResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
        String json = XML.toJson(RSSResponse.getEntity().getContent());
        return Response.ok().entity(json).build();
    }else{ // Handle Failure
        RSSResponse.getEntity().getContent().close();
        return Response.status(RSSResponse.getStatusLine().getStatusCode()).entity(RSSResponse.getStat
usLine().getReasonPhrase()).build();
    }
}
```

2. Create an SQL query and fetch the cities records from the database:

(getCitiesListJS adapter)

```
var getCitiesListStatement = WL.Server.createSQLStatement("select city, identifier, summary from weather;
");
function getCitiesList() {
    return WL.Server.invokeSQLStatement({
        preparedStatement : getCitiesListStatement,
        parameters : []
    });
}
```

(getCitiesListJava, getCitiesListJavaToJs adapters)

```
PreparedStatement getAllCities = conn.prepareStatement("select city, identifier, summary from weather");
ResultSet rs = getAllCities.executeQuery();
```

3. Loop through the cities records and fetch the weather info for each city from Yahoo! Weather Service:

(getCitiesListJS adapter)

```
for (var i = 0; i < cityList.resultSet.length; i++) {
   var yahooWeatherData = getCityWeather(cityList.resultSet[i].identifier);
...

function getCityWeather(woeid){
   return WL.Server.invokeProcedure({
      adapter : 'getCityWeatherJS',
      procedure : 'getYahooWeather',
      parameters : [woeid]
   });
}
```

(getCitiesListJava adapter)

```
while (rs.next()) {
  getWeatherInfoProcedureURL = "/getCityWeatherJava?cityId="+ URLEncoder.encode(rs.getString("identifi
er"), "UTF-8");
   HttpUriRequest req = new HttpGet(getWeatherInfoProcedureURL);
   HttpResponse response = adaptersAPI.executeAdapterRequest(req);
   JSONObject jsonWeather = adaptersAPI.getResponseAsJSON(response);
  ...
}
```

(getCitiesListJavaToJs adapter)

```
while (rs.next()) {
  // Calling a JavaScript HTTP adapter procedure
   HttpUriRequest req = adaptersAPI.createJavascriptAdapterRequest("getCityWeatherJS", "getYahooWeat
her", URLEncoder.encode(rs.getString("identifier"), "UTF-8"));
   org.apache.http.HttpResponse response = adaptersAPI.executeAdapterRequest(req);
   JSONObject jsonWeather = adaptersAPI.getResponseAsJSON(response);
  ...
}
```

4. Iterating through the retrieved rss feed to fetch only the weather description, put this values in a resultSet / JSONArray object and return it to the application:

(getCitiesListJS adapter)

```
...
if (yahooWeatherData.isSuccessful)
   cityList.resultSet[i].weather = yahooWeatherData.rss.channel.item.description;
}
return cityList;
```

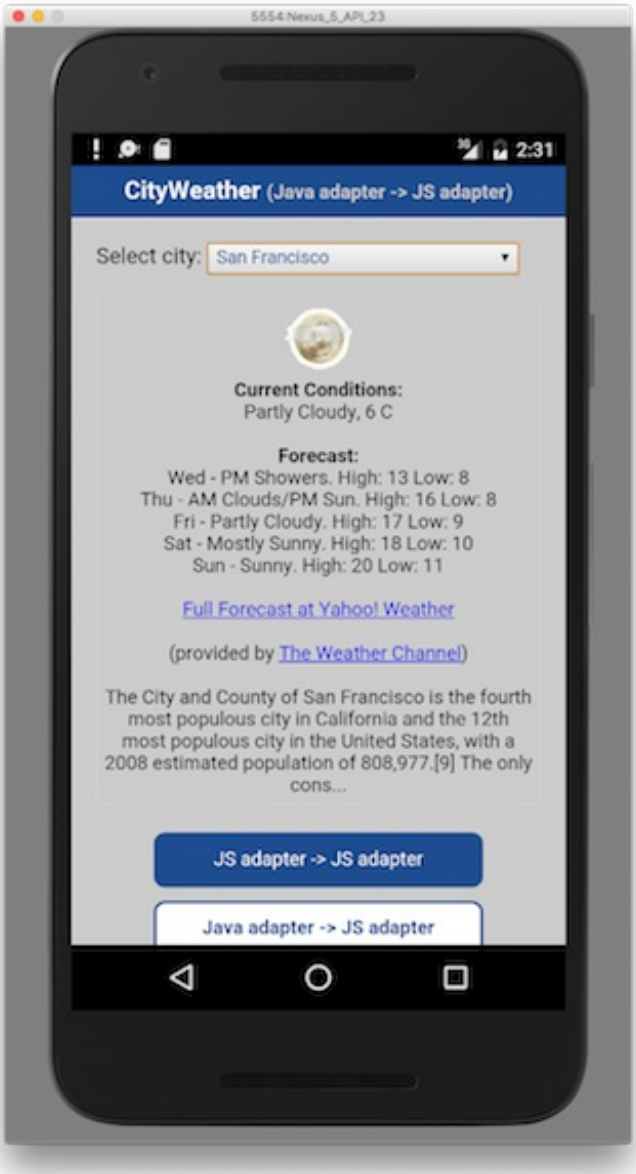
(getCitiesListJava, getCitiesListJavaToJs adapters)

```
{
    ...
    JSONObject rss = (JSONObject) jsonWeather.get("rss");
    JSONObject channel = (JSONObject) rss.get("channel");
    JSONObject item = (JSONObject) channel.get("item");
    String cityWeatherSummary = (String) item.get("description");

    JSONObject jsonObj = new JSONObject();
    jsonObj.put("city", rs.getString("city"));
    jsonObj.put("identifier", rs.getString("identifier"));
    jsonObj.put("summary", rs.getString("summary"));
    jsonObj.put("weather", cityWeatherSummary);

    jsonArr.add(jsonObj);
}
conn.close();
return jsonArr.toString();
```

Sample application

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

Note: the sample application's client-side is for Cordova applications, however the server-side code in the adapters applies to all platforms.



Sample usage

SQL setup

An example of city list in SQL is available in the provided adapter maven project (located inside the Cordova project), under `Utils/mobilefirstTraining.sql`.

1. Run the .sql script in your SQL database.
2. Use either Maven or MobileFirst Developer CLI to build and deploy the adapters (../../creating-adapters/).
3. Open the MobileFirst Console
    - Click on the **getCitiesListJS** adapter and update the database connectivity properties.
    - Click on the **getCitiesListJava** adapter and update the database connectivity properties.

Application setup

1. From the command line, navigate to the

project's root folder.

2. Add a platform by running the `cordova platform add` command.
3. Run the Cordova application by running the `cordova run` command.