Advanced Adapter Usage and Mashup

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.

This tutorial covers the following topics:

- 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 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

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(MyAdapterProcedureURL);
org.apache.http.HttpResponse response = api.getAdaptersAPI().executeAdapterRequest(req);
JSONObject jsonObj = api.getAdaptersAPI().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 = api.getAdaptersAPI().createJavascriptAdapterRequest(AdapterName, ProcedureNam
e, [parameters]);
org.apache.http.HttpResponse response = api.getAdaptersAPI().executeAdapterRequest(req);
JSONObject jsonObj = api.getAdaptersAPI().getResponseAsJSON(response);
```

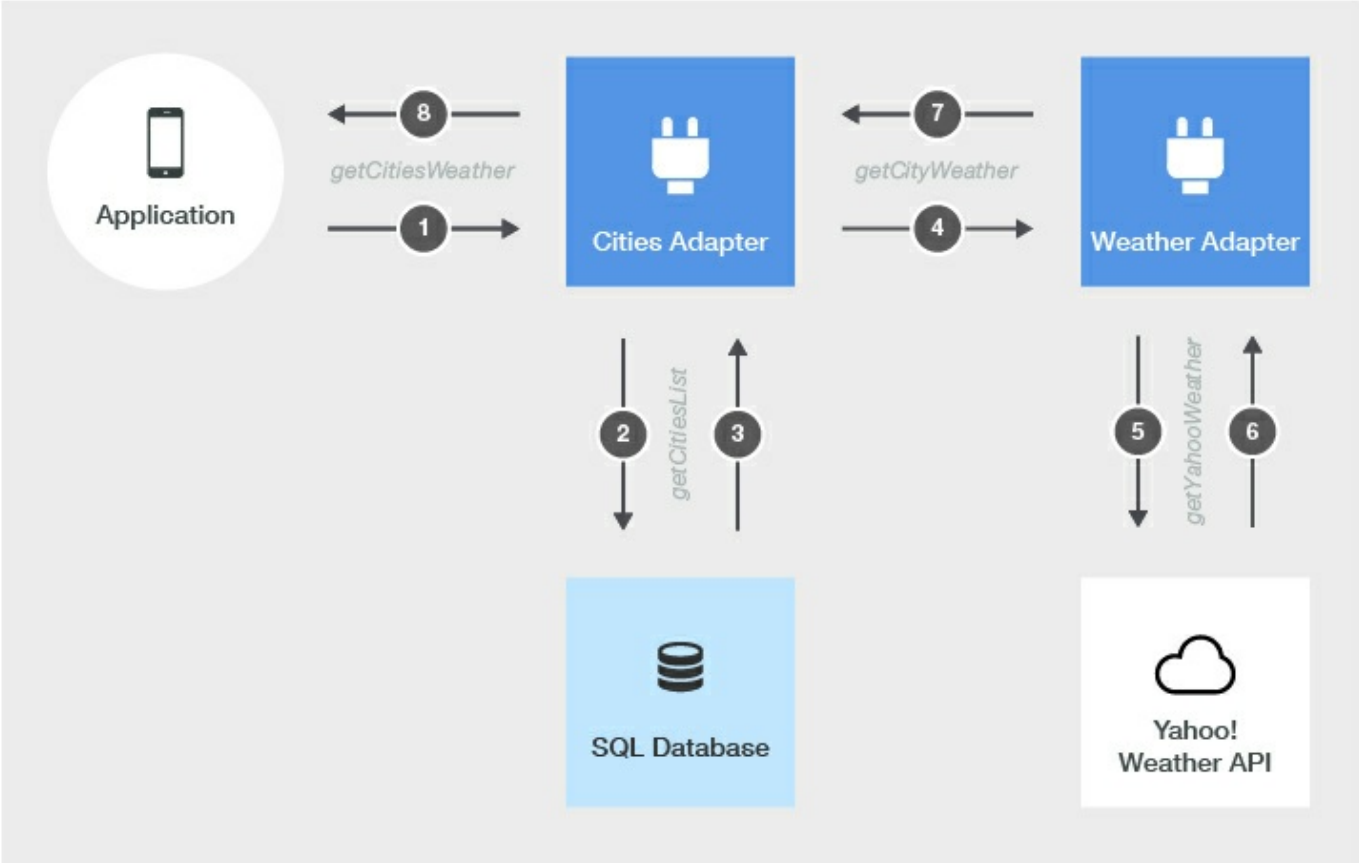
Data mashup example

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 String get(@Context HttpServletResponse response, @QueryParam("cityId") String cityId) throws C
lientProtocolException, IOException, IllegalStateException, SAXException {
    String returnValue = execute(new HttpGet("/forecastrss?w="+ cityId +"&u=c"), response);
    return returnValue;
}

private String execute(HttpUriRequest req, HttpServletResponse resultResponse) throws ClientProtocolEx
ception, IOException, IllegalStateException, SAXException {
    String strOut = null;
    HttpResponse RSSResponse = client.execute(host, req);
    ServletOutputStream os = resultResponse.getOutputStream();

...
// (Convert the retrieved XML to JSON)
}
```

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 = getSQLConnection().prepareStatement("select city, identifier, summary fr
om 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("ident
ifier"), "UTF-8");
   HttpUriRequest req = new HttpGet(getWeatherInfoProcedureURL);
   org.apache.http.HttpResponse response = api.getAdaptersAPI().executeAdapterRequest(req);
   JSONObject jsonWeather = api.getAdaptersAPI().getResponseAsJSON(response);
   ...
```

(getCitiesListJavaToJs adapter)

```
   while (rs.next()) {
      HttpUriRequest req = api.getAdaptersAPI().createJavascriptAdapterRequest("getCityWeatherJS", "get
YahooWeather", URLEncoder.encode(rs.getString("identifier"), "UTF-8"));
      org.apache.http.HttpResponse response = api.getAdaptersAPI().executeAdapterRequest(req);
      JSONObject jsonWeather = api.getAdaptersAPI().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)

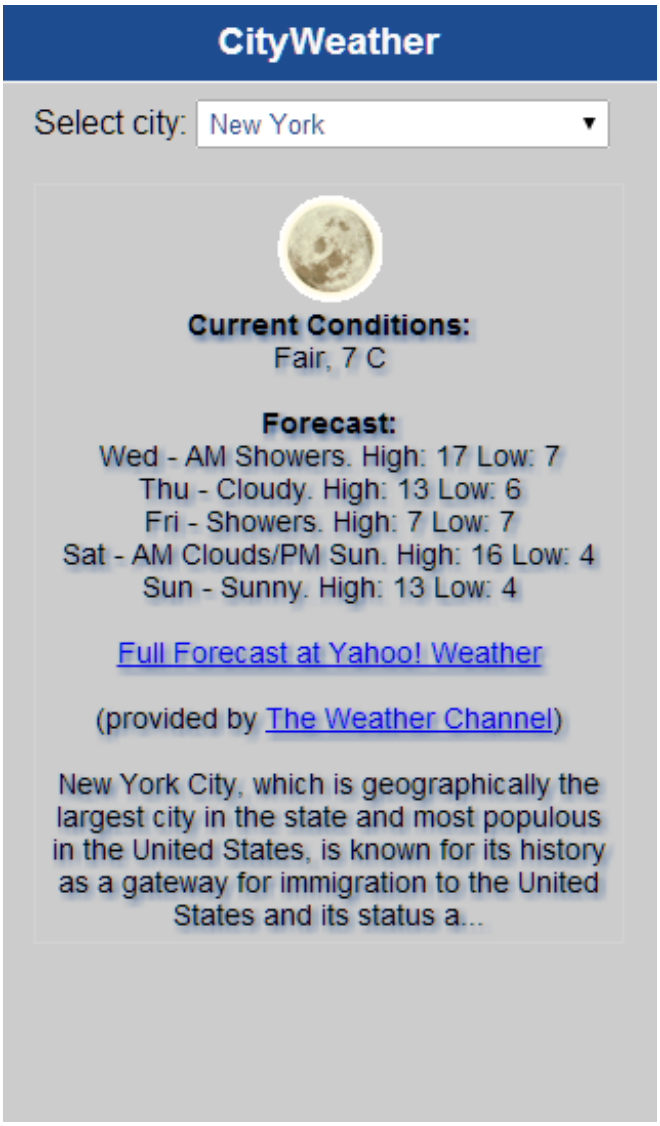
```
   JSONObject rss = (JSONObject) jsonWeather.get("rss");
   JSONObject channel = (JSONObject) rss.get("channel");
   JSONObject item = (JSONObject) channel.get("item");
   String cityWeatherSummary = (String) item.get("description");

   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();
```

> An example of city list in SQL is available in the provided adapter maven project, under `Utils/mobilefirstTraining.sql`. Remember that SQL adapters require a JDBC connector driver. Follow these instructions to add the JDBC connector dependency (https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html) .

Sample application

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

1. Use either Maven or MobileFirst Developer CLI to build and deploy the adapter (../../creating-adapters/).
2. From the command line, navigate to the Cordova project.
3. Add a platform by running the `cordova platform add` command.
4. Run the Cordova application by running the `cordova run` command.