

# Advanced adapter usage and mashup

fork and edit tutorial (<https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/#fork-destination-box>) | [report issue](#)

(<https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/issues/new>)

- Download Studio project

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/AdapterMashUpProject.zip>)

## Overview

Now that basic usage of different types of adapters has been covered in previous tutorials, 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, one could make several requests successively, one depending on the other. However, writing this logic on the server-side could be faster and cleaner.

## invokeProcedure

At the heart of this mashup scenario is the `WL.Server.invokeProcedure(invocationData)` API. Almost identical to its client-side counterpart, it enables you to invoke a procedure on any of your adapters.

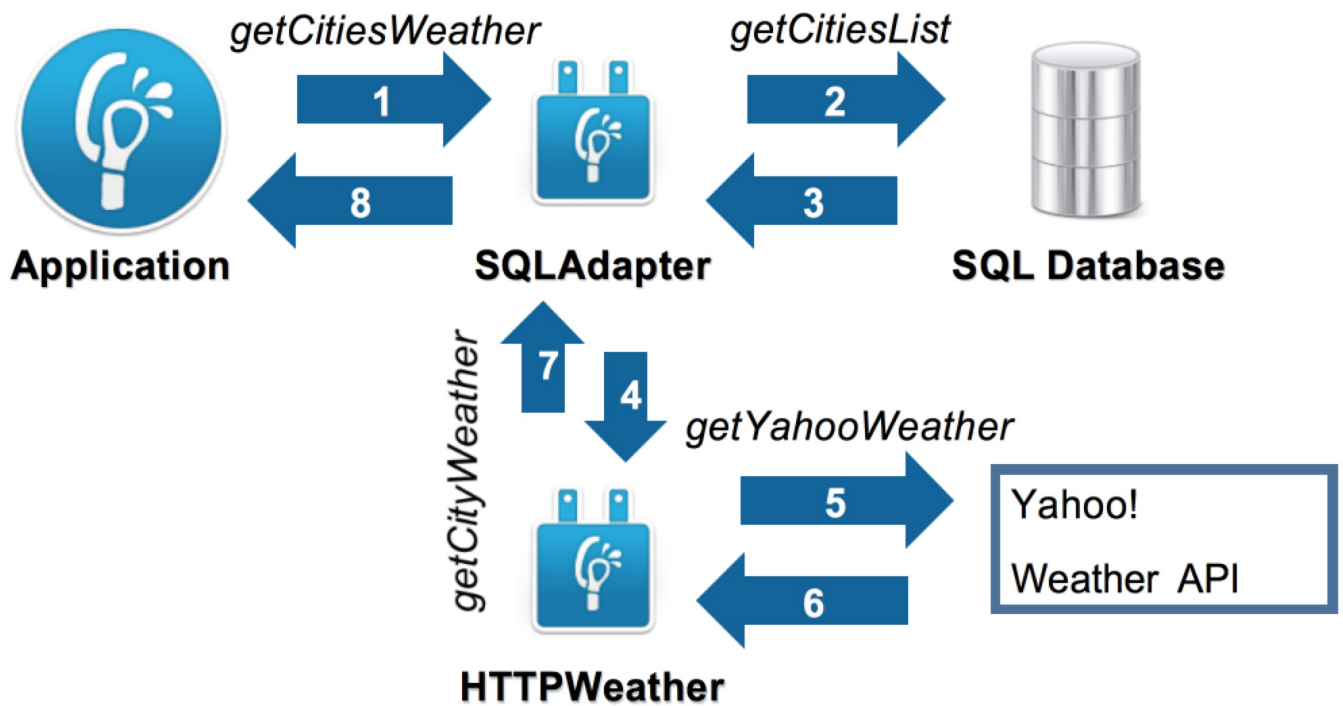
The `invocationData` parameter has the same format as the client-side API.

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

However, while the client-side `invokeProcedure` uses a success handler, the server-side one returns the result object itself.

## Data Mashup Sample

The following example shows how to mash up data from two data sources and to return the data stream to the application as a single `invocationResult` object.



Data mashup can be implemented in the same way for any number of data sources and across different adapter types.

As an example, mash up data from the following sources:

- SQL:
  - 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.
- HTTP:
  - Connect to the Yahoo! Weather Service.
  - Extract an updated weather forecast for each of the cities that are retrieved via SQL.

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

## HTTP Adapter

1. Create an HTTP adapter and name it **HTTPWeather**. This adapter connects to Yahoo! Weather RSS feed at <http://weather.yahooapis.com/forecastrss>.  
The adapter has a single procedure called `getYahooWeather`.  
The HTTP request has the following parameters:
  - `w` – Where in the world ID, a city ID used by Yahoo!
  - `u` – units. Can be "c" for Celsius or "f" for Fahrenheit.
2. Create the `getYahooWeather` function in the **HTTPWeather-impl.js** file and use it to retrieve and return the weather data by using the Yahoo! Weather API:

```

function getYahooWeather(woeid) {
  var input = {
    method : 'get',
    returnedContentType : 'xml',
    path : 'forecastrss',
    parameters : {
      'w' : woeid,
      'u' : 'c' //celsius
    }
  };
  return WL.Server.invokeHttp(input);
}

```

## SQL Adapter

1. Create a SQL adapter and name it **SQLAdapter**. This adapter contains:
  - **getCitiesWeather** – a public procedure, which is declared in the adapter XML file. It is called from the application and returns the mashed-up data.
  - **getCitiesList** – an internal (private) function, which is not declared in the adapter XML file. It is called from the adapter through server API to get the cities list from the SQL server.
  - **getCityWeather** – an internal (private) function, which is not declared in the adapter XML file. It calls the **getYahooWeather** procedure from a different adapter (HTTP) with the Yahoo! Weather ID as a parameter.

An example of city list in SQL is provided with the attached sample, under **server/mobilefirstTraining.sql**.

Remember that SQL Adapters require a JDBC connector driver, which must be downloaded separately by the developer and added to the **server/lib** folder of the project.

2. Create a **getCitiesList** function in the **SQLAdapter-impl.js** file and use it to retrieve and return the city list from the SQL database.

```

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

```

3. Create a **getCityWeather** function in the **SQLAdapter-impl.js** file and use it to retrieve and return the weather data from the HTTPWeather adapter:

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

4. In the SQL adapter, extract a city identifier from the city list that was retrieved earlier and use it to call the `getYahooWeather` procedure of the HTTPWeather adapter.
5. Repeat for each city, attaching the received data to the city object.

```
function getCitiesWeather(){
  var cityList = getCitiesList();
  for (var i = 0; i < cityList.resultSet.length; i++) {
    var yahooWeatherData = getCityWeather(cityList.resultSet[i].identifier);
    if (yahooWeatherData.isSuccessful)
      cityList.resultSet[i].weather = yahooWeatherData.rss.channel.item.description
  }
  return cityList;
}
```

6. Return the mashed-up data to the application, which you can process in client-side code.

## Sample application

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/AdapterMashUpProject.zip>) the Studio project.

## CityWeather

Select city:  ▼



### Current Conditions:

Fair, 7 C

### Forecast:

Wed - AM Showers. High: 17 Low: 7  
Thu - Cloudy. High: 13 Low: 6  
Fri - Showers. High: 7 Low: 7  
Sat - AM Clouds/PM Sun. High: 16 Low: 4  
Sun - Sunny. High: 13 Low: 4

[Full Forecast at Yahoo! Weather](#)

(provided by [The Weather Channel](#))

New York City, which is geographically the largest city in the state and most populous in the United States, is known for its history as a gateway for immigration to the United States and its status a...