Getting started with mobile development for z/VSE

Last formatted on: Wednesday, February 25, 2015

Alina Glodowski

alina.glodowski@de.ibm.com

zvse@de.ibm.com

Contents

Introduction

Industries of all varieties have begun to realize that the target audiences for their business applications have shifted in massive numbers from the use of traditional personal computers, such as desktops and laptops, to using mobile devices such as smart phones and tablets for accessing the internet and for obtaining the information they seek. This applies if the intended audience for the application is a direct customer of the enterprise (Business-to-Consumer apps, or "B2C"), or if the targeted user is an employee or business partner ("B2E" and "B2B", or Business-to-Employee and Business-to-Business apps). Across the globe, more people are now using mobile devices that they can carry with them wherever they go, and which are more user-friendly and intuitive to use, as their primary means of obtaining information and requesting services over the internet.

IBM® MobileFirst Platform Foundation, formerly known as IBM Worklight, helps organizations extend their business to mobile devices. It provides an open and comprehensive platform to not only build, but test, run and manage native, hybrid and mobile web apps. Available as an on premises or private cloud solution, IBM MobileFirst Foundation can help reduce both application development and maintenance costs, improve time-to-market and enhance mobile application governance and security. [1]

IBM MobileFirst Foundation is comprised of five components:

1. **IBM MobileFirst Studio** offers leading tools for mobile app development that help maximize code reuse and accelerate development.
2. **IBM MobileFirst Server** is mobile-optimized middleware that serves as a gateway between applications, back-end systems and cloud-based services.
3. **IBM MobileFirst Device Runtime Components** offer runtime client application program interfaces (API) designed to enhance security, governance and usability.
4. **IBM MobileFirst Application Center** enables you to set up an enterprise app store that manages the distribution of production-ready mobile apps.
5. **IBM MobileFirst Console** is an administrative GUI designed to provide real-time operational analytics for the server, adapters, and applications and push services to help you manage, monitor and instrument mobile apps.

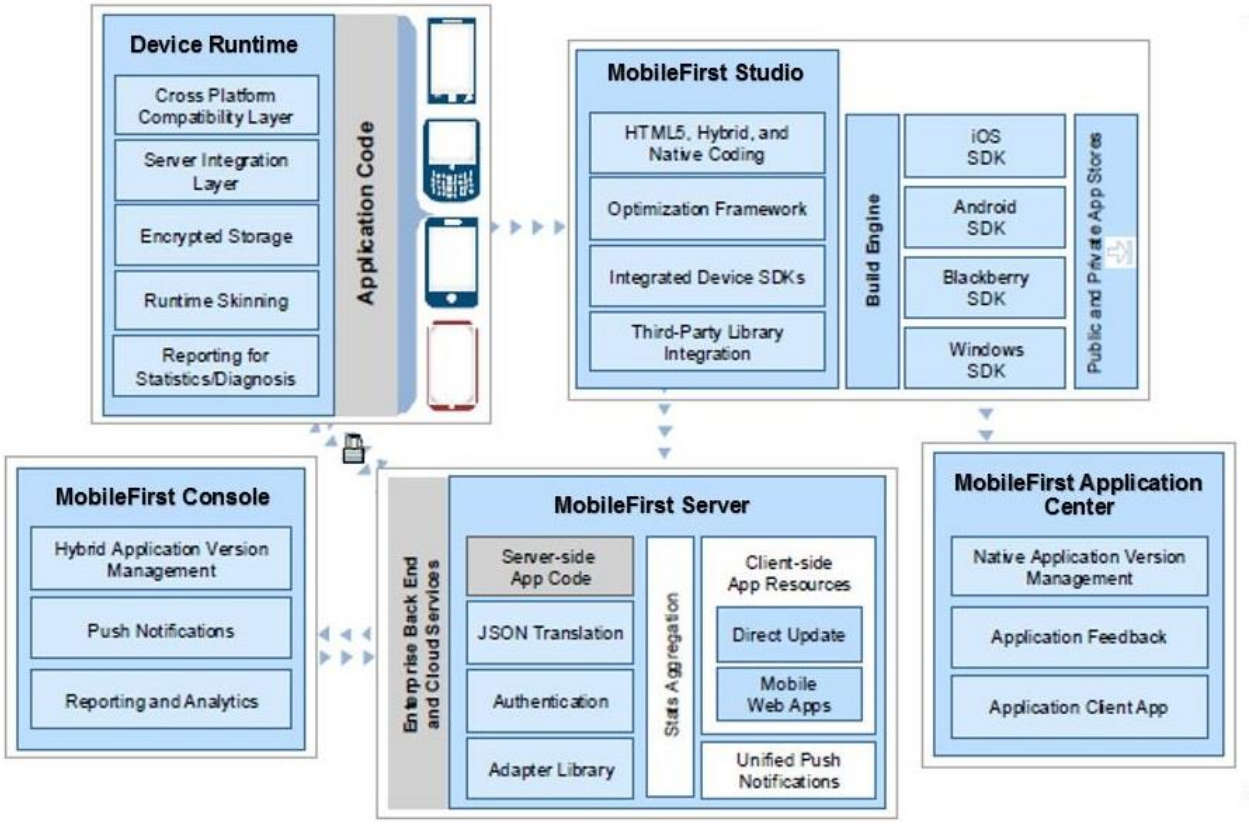

Figure 1. The five components of IBM MobileFirst

With IBM MobileFirst Foundation you can:

- Build apps for any mobile operating environment and device with your preferred development approach – native, hybrid or mobile web.
- Connect and synchronize mobile apps with enterprise data, applications and cloud services, including IBM BlueMix™.
- Safeguard mobile security at the device, application, data and network layer.
- Manage your mobile app portfolio from a single central interface with detailed operational analytics.

Overview

This paper demonstrates that z/VSE has all necessary tools to provide support for MobileFirst development.

As shown in the Figure 2, z/VSE has a set of tools and components to provide different sort of connections between z/VSE and external services.

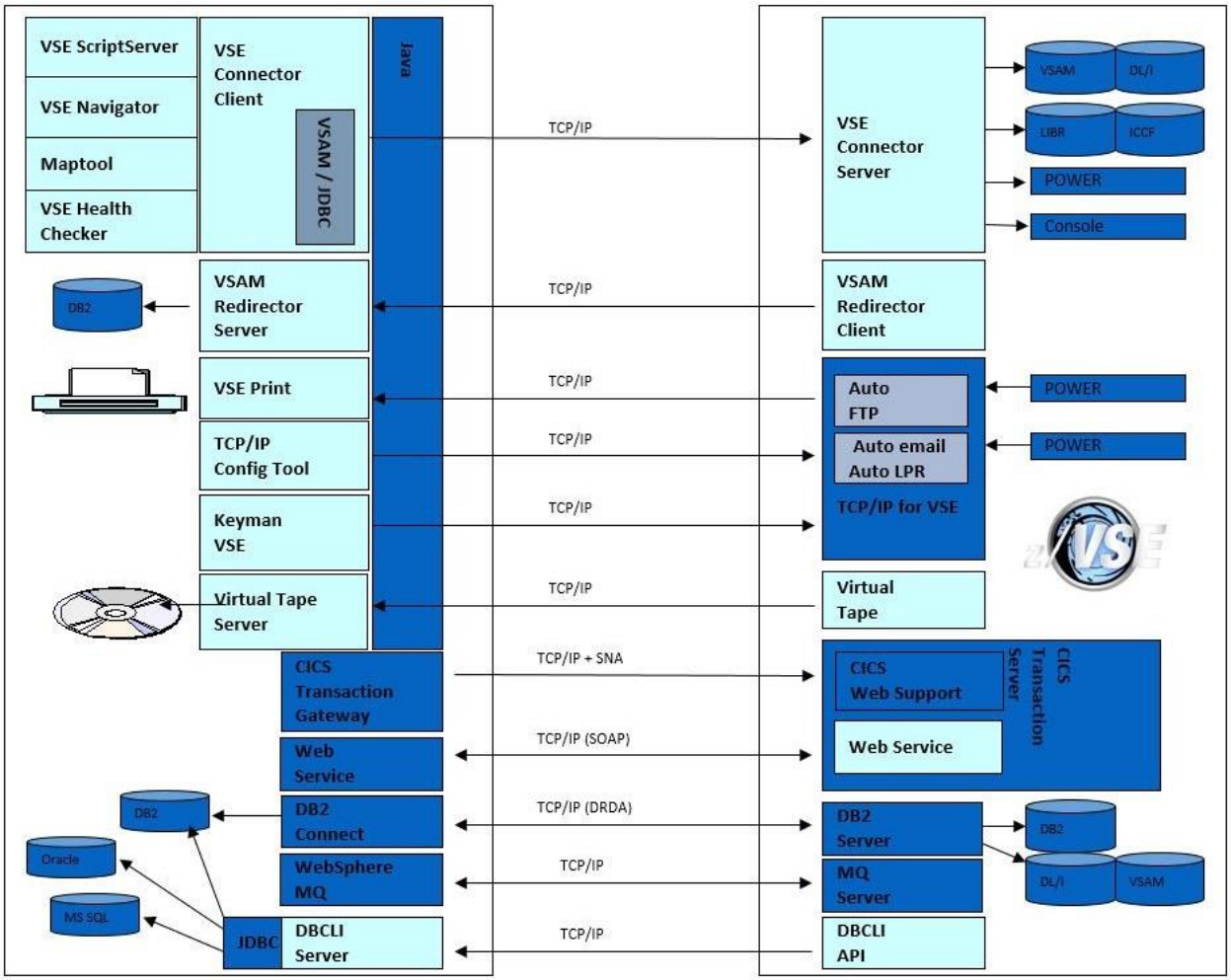

Figure 2. z/VSE Connectors Overview

In this paper will be shown how to use z/VSE Connector Client/Server and Web Services for the mobile development. The process of creating mobile User Interface and mobile Logic will also be demonstrated shortly.

To start mobile development with z/VSE, you need to have the following applications:

- The **IBM MobileFirst Platform Developer Edition**, formerly known as IBM Worklight Developer Edition. It is a collection of self-contained, easy-to-install development tools for IBM MobileFirst Platform Foundation, including: MobileFirst Studio, MobileFirst Test Workbench, and MobileFirst Command Line Interface. [2]
- The **z/VSE Connector Client** (can be downloaded from the web-page [3]) provides the z/VSE Java Beans class library, online documentation and programming reference (JavaDoc), and many samples including Java source code for writing Web applications such as applets, servlets, Enterprise Java Beans (EJBs), etc.
- The **z/VSE Connector Server** (part of VSE/ESA 2.5 and later releases) is running on z/VSE and implements native access methods to VSE/VSAM, Librarian, VSE/POWER, ICCF (read-only), allowing you to submit jobs and access the z/VSE operator console.

The last two, z/VSE Connector Client and z/VSE Connector Server, are needed only if you want to have more flexible and powerful access to your z/VSE data.

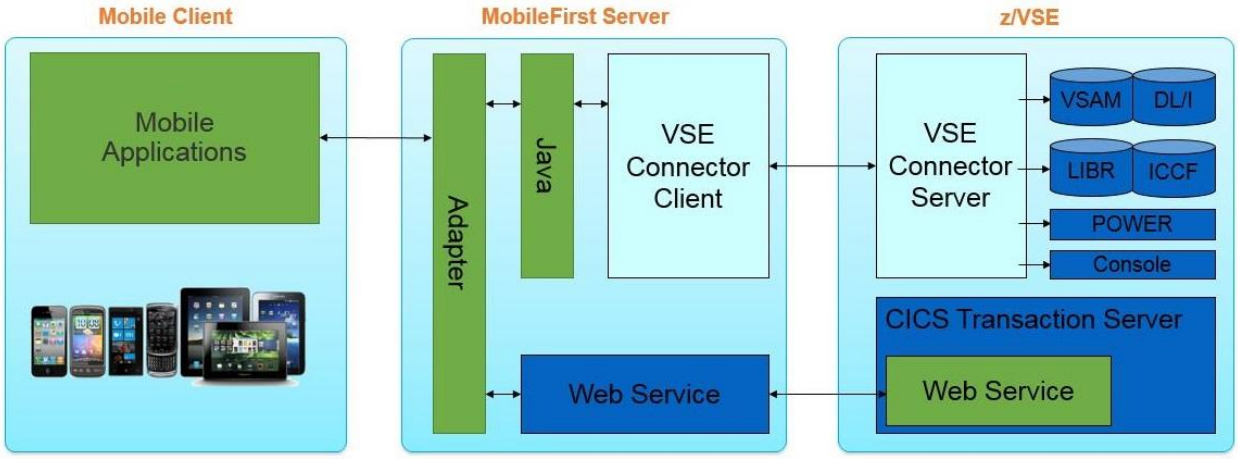

Figure 3. MobileFirst components and z/VSE project structure

With z/VSE Connectors component you can:

- Use Web Services in your Mobile environment
- Use VSE Connector Client and VSE Connector Server to gain access to your data under z/VSE from your mobile device

As soon as you have installed needed applications from the list above, you can start mobile development for z/VSE. Development is performed for the parts marked green in the Figure 3. You must implement an adapter as a middle layer which knows where to connect to, which parameters to pass, and what expect as a result.

Two types of adapters can be used with z/VSE:

1. SOAP/HTTP Adapter. It works as Web Service using SOAP Engine and CICS Transaction Server on z/VSE.
2. HTTP Adapter together with Connector Client library. With it, more development is supposed to be done. The advantage is that it gives more flexible and powerful access to your z/VSE data.

This paper describes both the adapters and relevant approaches.

Web Services adapter

Let's assume you have a working CICS Application with running SOAP Service for it. If you do not have SOAP Service, please refer to the document [4] for information on how to start with it.

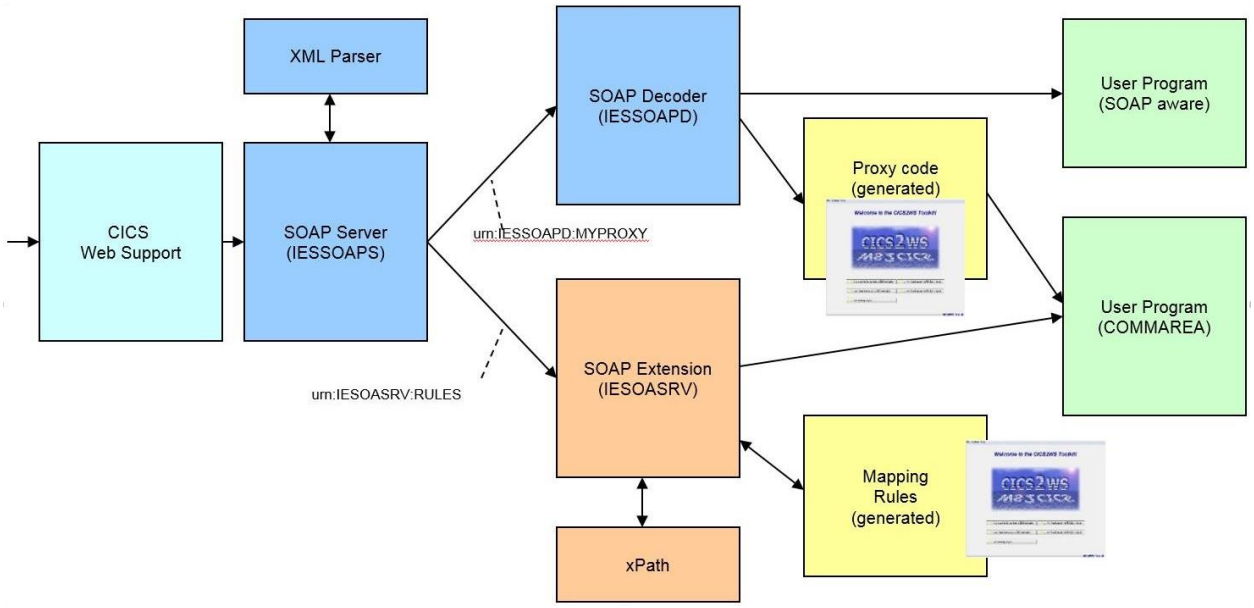

Figure 4. Overview of Web Services within z/VSE

The following data will be used in z/VSE in demonstration purposes:

1. User program *FFSTIO* works with VSAM data *FFSTORE*
2. *FFSTORE* file content has a list of cafes with their addresses. The key field is *storeID* (index number in VSAM cluster).
3. Copybook *FFSTIO.cb* is used for comarea.

```
          03 FSTIO-MAP.
                   05 ACTION              PIC 9(8) COMP.
                   05 RETURN-CODE         PIC 9(8) COMP.
                   05 FILE-NAME           PIC X(8).
                   05 STORE-ID            PIC X(6).
                   05 STORE-NAME          PIC X(25).
                   05 LOC-STREET          PIC X(25).
```

```
                05 LOC-CITY           PIC X(25).
                05 LOC-ZIP            PIC X(10).
                05 LOC-COUNTRY        PIC X(25).
                05 LOC-REP            PIC X(20).
                05 VAL1               PIC 9(8) COMP.
                05 VAL2               PIC 9(8) COMP.
                05 DATE               PIC X(10).
                05 WEB-PIC1           PIC X(20).
                05 WEB-PIC2           PIC X(20).
                05 A-CODE             PIC X(10).
                05 FILLER             PIC X(6).
```

Based on this data, CICS2WS Tool [11] creates a wsdl file. It will be used for communication between the user program and the mobile application. The following parameters are defined:

- Input parameters: ACTION, FILE-NAME, STORE-ID
- Output parameters: RETURN-CODE, STORE-NAME

Goal is to get name of the store based on the store ID.

At first, an adapter will be created. This adapter will implement web-service. As the second step, a simple mobile application that will call the created adapter will be developed.

Creating a project

Create a new MobileFirst Project with Hybrid Mobile App (please read [9] about different types of mobile app). In Eclipse, choose File → New → MobileFirst Project. Detailed information on how to create a mobile application with MobileFirst Platform can be found at [10].

After that, you will get the following folder structure in your Workspace:

Figure 5. Project structure

Here:

- Name of the project: **ffStorePrj**;
- Name of the mobile application: **ffStoreApp**.

In the Figure 6 below, you can find more detailed information how projects folder structure is related to the MobileFirst components. In addition to the information above, you can see there two adapters that will be created in the next section:

- **FFStoreAdapter** – pure HTTP adapter;
- **SoapAdapter1** – automatically created adapter for Web Service.

In the folder <*your project name*> → service you can also see an integrated service for the SoapAdapter1.

Let's have a closer look to the adapters.

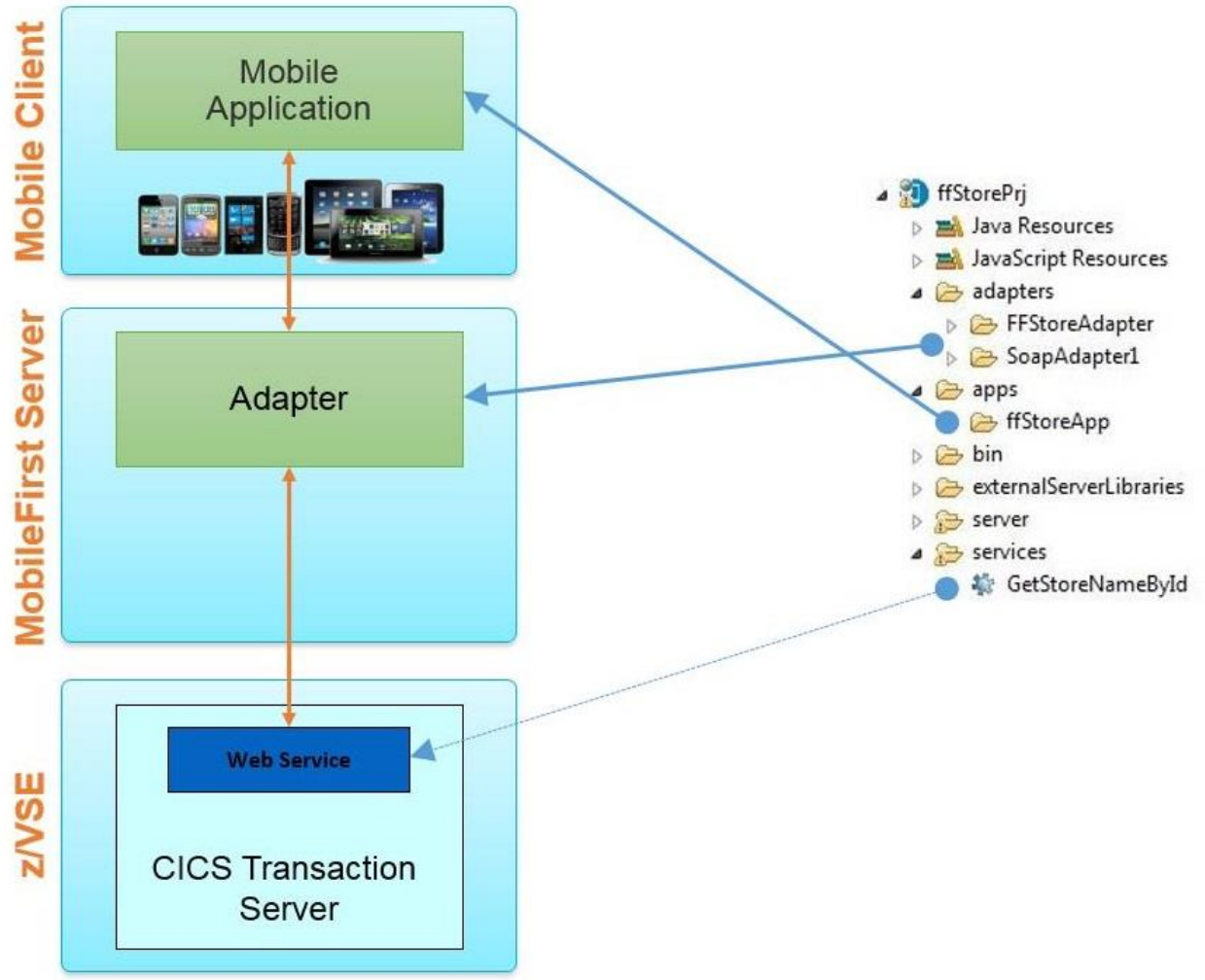

Figure 6. Relationship between project folders and MobileFirst structure

Creating an adapter

There is a choice here: you can create a SOAP Adapter or pure HTTP Adapter. Let's have a look at both variants.

SOAP adapter

- Left click on <*your project name*> → services, find "Discover Back-end Services"
- Choose "Web Services Definition Language"

- Find your wsdl file and choose the function you want to use

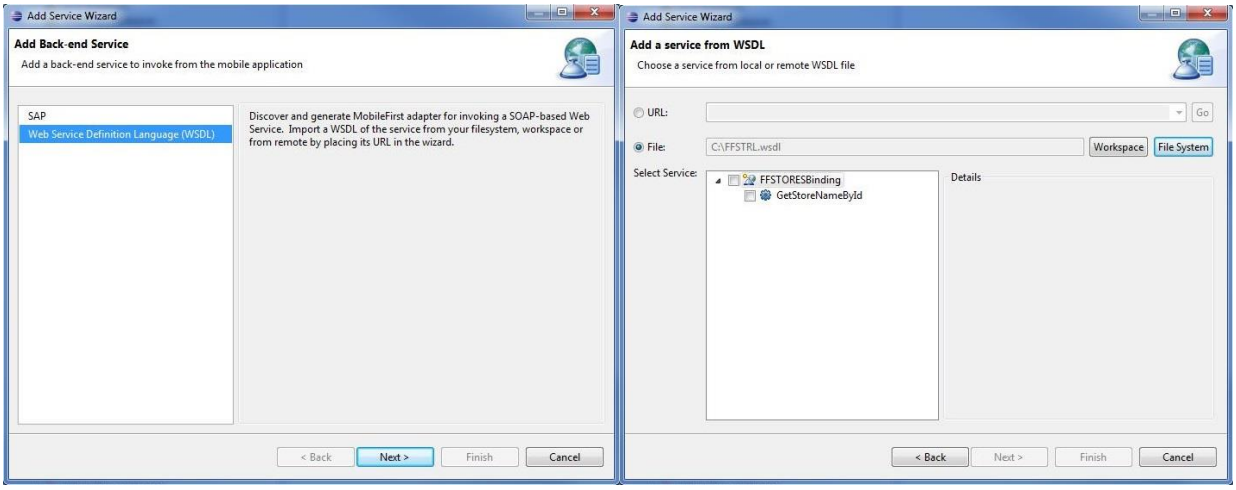

Figure 7. Create an adapter based on wsdl-file

By default, adapter "SOAPAdapter1" will be created. If you do not like this name, you can rename the folder and files inside. In this sample, wsdl file from the beginning of this chapter is used. The name of service is GetStoreNameById.

Note that MobileFirst automatically creates adapter based on JSON format.

HTTP adapter

Select <*your project name*> → adapters + right click → New → MobileFirst Adapter.

In this sample name **FFStoreAdapter** is used. You can specify any name you prefer.

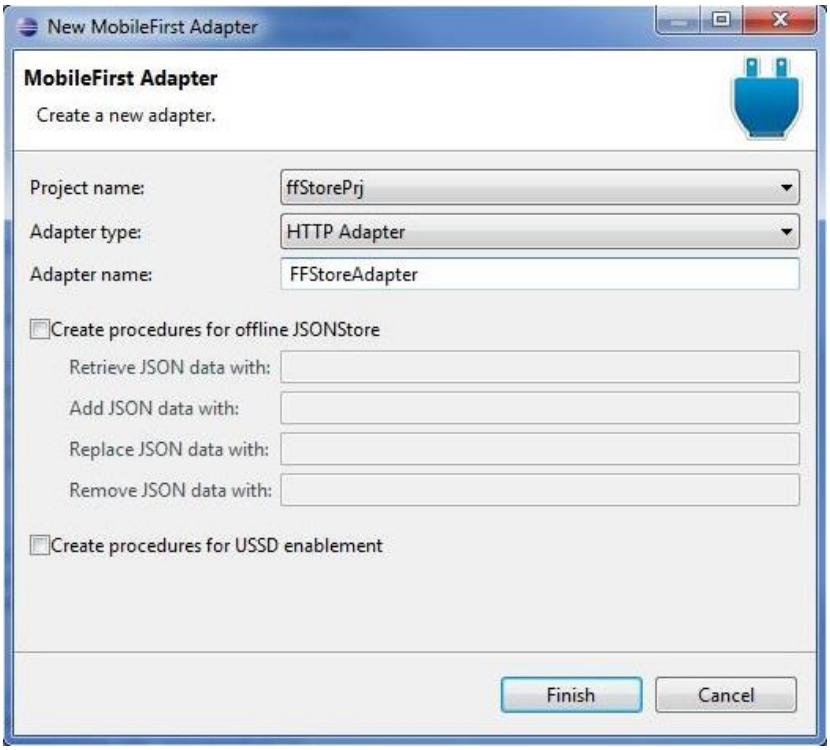

Figure 8. Create a simple adapter

MobileFirst will automatically create a simple adapter for you. Change it in the following way:

- <*your project name*> → adapters → filtered.xsl
  - Delete this file
- <*your project name*> → adapters → <*your adapter name*>.xml
  - Add procedure *getStoreName*
  - Delete procedures *getStories* and *getStoriesFiltered*
  - Change 'Connectivity': add z/VSE host and port
- <*your project name*> → adapters → <*your adapter name*>-impl.xml
  - Delete functions *getStories*, *getStoriesFiltered*, *getPath*
  - Add the following code:

```
function getStoreName(params){

      var body = '<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" ';
```

```
        body += 'xmlns:urn="urn:IESOASRV:FFSTRL"> ';
        body += '<soapenv:Header/>';
        body += '<soapenv:Body>';
        body += '<urn:GetStoreNameById>';
        body += '<ACTION>1</ACTION>';
        body += '<FILENAME>FFSTORE</FILENAME>';
        body += '<STOREID>' + params + '</STOREID>';
        body += '</urn:GetStoreNameById>';
        body += '</soapenv:Body>' + '</soapenv:Envelope>';

        var input = {
              method : 'post',
              returnedContentType : 'xml',
              path : '/cics/CWBA/IESSOAPS',
              body: {
                  content : body.toString(),
                  contentType : 'text/xml; charset=utf-8'
              }
          };

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

Note that two of the three incoming parameters were hard coded in adapter:

```
'<ACTION>1</ACTION>'
'<FILENAME>FFSTORE</FILENAME>'
'<STOREID>' + params + '</STOREID>'
```

It was done only to simplify the code. You can pass as many parameters as you want.

Note that here JSON format is not used. You need to control yourself how to pass your parameters between mobile application and MobileFirst adapter.

Now you can test the created adapter. Right click on Adapter folder, choose Run As → Invoke MobileFirst Procedure.

Invocation Result of procedure: 'getStoreName' from the MobileFirst Server:


```
{
   "Envelope": {
      "Body": {
         "GetStoreNameByIdResponse": {
            "RETURNCODE": "0",
            "STORENAME": "Frechdax"
         }
      },
      "soapenv": "http:\/\/schemas.xmlsoap.org\/soap\/envelope\/",
      "urn": "urn:IESOASRV:FFSTRL"
   },
   "errors": [
   ],
   "info": [
   ],
   "isSuccessful": true,
   "responseHeaders": {
      "Content-Length": "00000292",
      "Content-type": "text\/xml; charset=utf-8"
   },
   "responseTime": 99,
   "statusCode": 200,
   "statusReason": "OK",
   "totalTime": 100,
   "warnings": [
   ]
}
```


Figure 9. Invocation result

If you see the picture above in your browser, this means your adapter works with your z/VSE application. Otherwise, you will get another status code and/or status message at this moment. The following reason could be indicated:

- No connection
- No program in z/VSE
- Not correct request
- etc

To solve the problem, you need to check carefully invocation result and, if necessary, turn on tracing in z/VSE.

Creating mobile application

After you have checked that the adapter is working properly, you can start creating mobile application. In this sample, a very simple Application will be created, it consists of the two files: User Interface (index.html) and business logic (main.js).

<your Project Name> → apps → *<your App name>* → common → index.html

- Add header
- Add label and input field
- Add button to get results
- Add output field

```
<div data-role="page" id="page">
	<div data-role="header" id="header" data-position="fixed">
		<h3>FFStore App</h3>
	</div>
	<div data-role="content" style="padding: 15px">
		<!--application UI goes here-->
		<label for="text">Type Store ID:</label> <input type="number"
		name="storeID" id="storeID" value="00001">
		<a href="#" data-role="button" onclick="getStoreNameById();" id="button"
		data-icon="forward">Get Store Name</a>
		<label id="storeName">Result:</label>
	</div>
</div>
```

Upon clicking on the button “Get Store Name”, mobile application will call getStoreNameById() function (that was not created yet). So, you need to add the following code into *<your Project Name>* → apps → *<your App name>* → common → js → main.js file:

```
function getStoreNameById( ){

	// Get input parameters
	var storeId = document.getElementById('storeID').value;

	// Predefine adapter data
	var invocationData = {
			adapter : 'FFStoreAdapter',
			procedure : 'getStoreName',
			parameters : [storeId]
		};

	// Call adapter and show results
	WL.Client.invokeProcedure(invocationData,{
		onSuccess : showResultSuccess,
		onFailure : showResultFailure
	});
}

// Successful connection, not necessary successful result
function showResultSuccess(result){
	alert("Connection was successful");
}

// Not successful connection
function showResultFailure(result){
	alert("Connection fails");
}
```

Note that there is only one parameter (*storeId*) in the *invocationData*, *since* two of the three parameters were hard coded in the adapter. In case you use automatically created **SoapAdpater1**, you need to pass here parameters in the right format and order.

Function **getStoreNameById**(), during its execution, will connect to the MobileFirst Server, search for adapter **'FFStoreAdapter'**, and call procedure **'getStoreName'** with parameter *storeId*. After the connection has been established, it will receive one of the following names: either function **showResultSuccess**, if the connection was successful, or function **showResultFailure**, otherwise.

Now you can test your application.

Select <*your project name*> → apps → <*your app name*> + right click → Run as → Preview.

In case of successful connection, you will see the following message:

Figure 10. Successful Connection

Otherwise, you will see the "Connection fails" message.

Your user interface will not be changed to see a result that you get from Web Service, you need to modify function **showResultSuccess()** to view it.

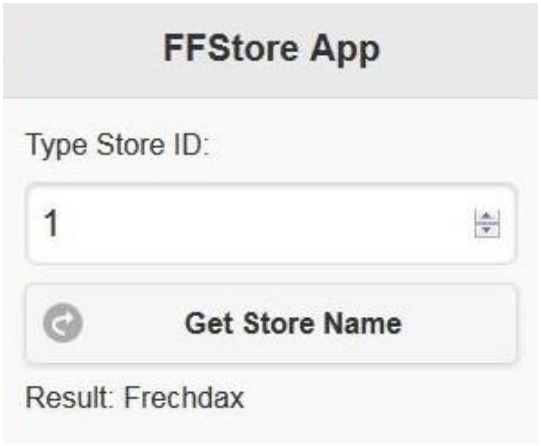

Figure 11. Result received from z/VSE

For debugging your mobile application, you can use a build-in Web Development Tools in your browser.

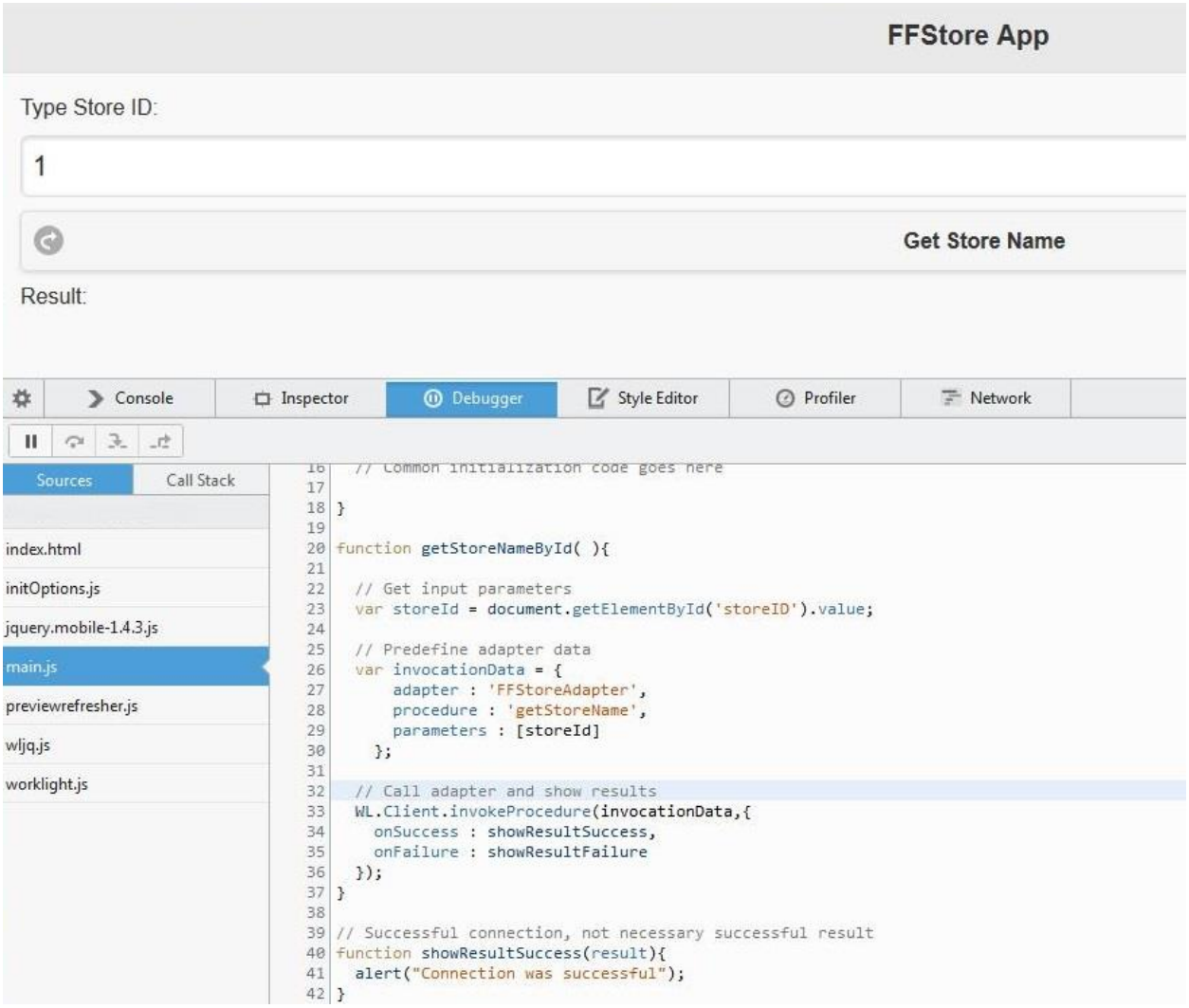

Figure 12. Debugging application

z/VSE Connectors adapter

Let's consider a sample for the second adapter.

There is a skeleton SKVSSAMP in library 59. This skeleton creates a file (in VSAM KSDS format) and fills it with some data. You can change skeleton and data if you want. By default, after running the skeleton you will get the following data on you z/VSE:

Export of a VSAM data in HTML format

Catalog: VSESP.USER.CATALOG
Cluster: VSAM.CONN.SAMPLE.DATA
Map: USEDCARS
Number of records: 7
Date: 23.12.2014 10:38:23

ARTICLENO	MANUFACTURER	TYPE	MODEL	HP	DISPLACEMENT	CYLINDERS	COLOUR	FEATURES	PRICE
1	Volkswagen	New Beetle	Petro Model	115	2000	4	Red	Sliding Roof	17000
2	Mustang	GT 2	DR CONV	250	4600	8	Black	Smoker's Package	30190
3	Ford	Taurus	SE Station Wagon	200	3000	6	Blue	Appearance Package	23280
4	BMW	compact	316i	102	1600	4	VelvetBlue Metallic	sport edition	20500
5	Mercedes	E220	Avantgarde	160	2300	6	Grey	Navigation System	67000
6	Porsche	Roadster		220	2700	6	Silver	Leather,CD Changer	42300
7	Ford	Escort	ZX2 2 Door Coupe	150	2000	6	White	Sp.Seats,Zetec Eng.	15715

Figure 13. VSAM data Sample

To have more flexible access to your data, you can use Connector Server (on z/VSE) and Connector Client as an external library for your adapter.

First, you need to download Connector Client from z/VSE web-page [12] and install it at any folder you want.

After the installation, you will see the following set of files and folders:

- _uninstallConn
- conndoc
- samples
- cci.jar
- ConnectorReadme.txt
- ibmjsse.jar
- ibmpkcs.jar

- VSEConnector.jar
- VSEConnectors.html

VSEConnectors.html together with content of *conndoc* folder provide full information about API for the z/VSE Connectors. There is a big variety of functions for using.

Folder samples contain different code examples to use z/VSE connectors.

To use connectors, you need to add VSEConnector.jar, cci.jar, ibmjsse.jar, ibmpkcs.jar to your mobile project. Just copy these libraries to <*your project name*> → server → lib.

Your java source code for adapter will be located under <*your project name*> → server → java.

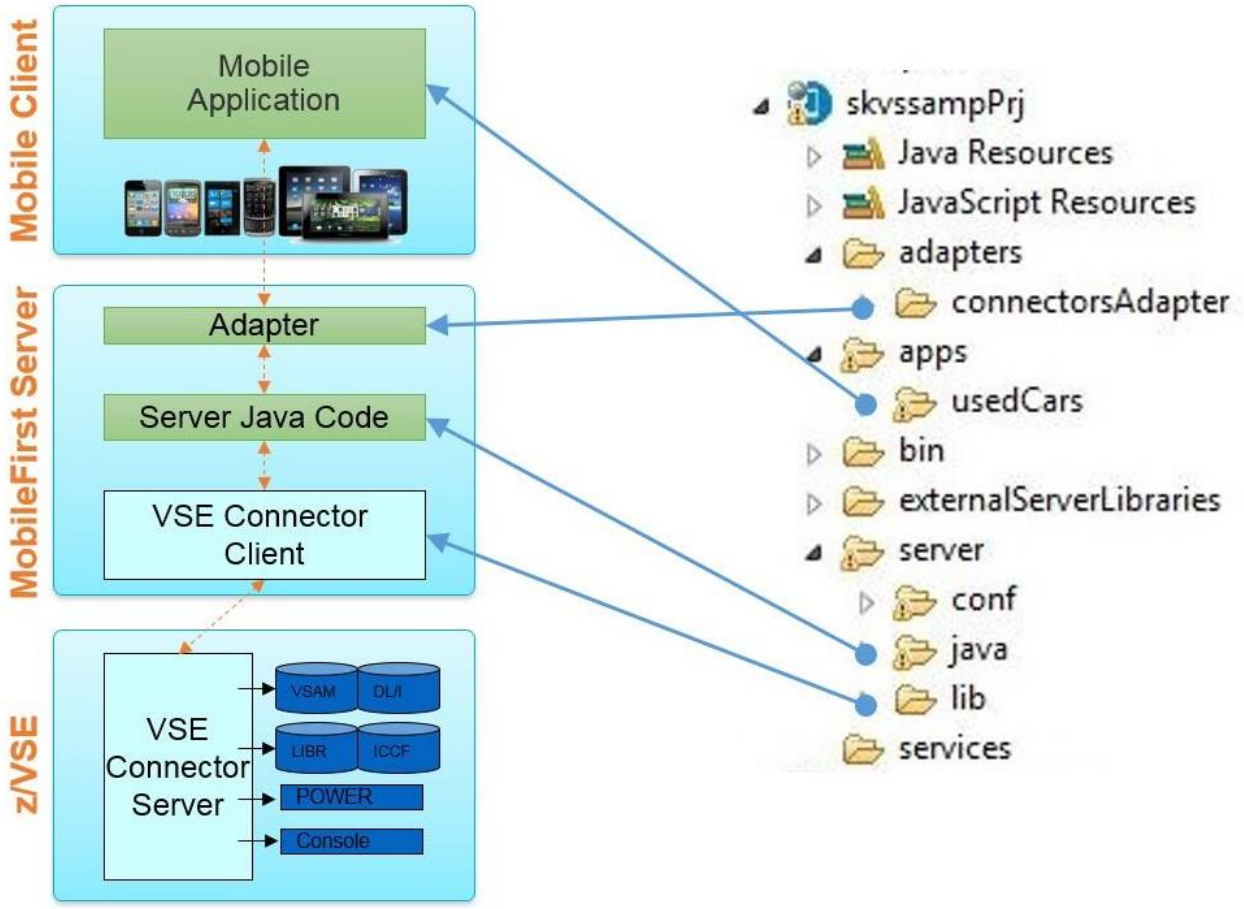

Figure 14. Project structure and relationship to MobileFirst components

In the same way as with SOAP/HTTP adapter, you need to create a new project. In this sample, the name skvssampPrj will be used. Mobile application will be called usedCar. You can choose any name you prefer. The project structure is shown in Figure 14:

- server/lib folder has z/VSE Connectors libraries;
- server/java folder will have your Java code.

Parts marked green should be developed.

Our next steps will be:

- Create an Mobile App and Adapter that connect to z/VSE to the defined cluster
- Enable the application to read records, add new records, and modify records including deletion.

MobileFirst server Java code and adapter

Before starting to create adapter, you need to implement necessary functions such as *"get VSAM records"*, *"add VSAM record"*, and *"modify VSAM record"*. Use provided connectors sample to implement them. Code for these functions could be found in [5]. This java code should be located under <*your project name*> → server → java.

In this sample, main java file is called *skvssampJava.java* with the predefined package *com.ibm.zvse.adapter*. This is important information for the adapter.

Note that JavaScript on Mobile expects to receive data from adapter in JSON format. For this, it's necessary to add the function that will convert data into JSON format.

To create an adapter, choose:

<*your project name*> → adapter + right mouse click → New → MobileFirst Adapter

Simple http adapter with predefined function will be created. Make the following changes:

- <*your project name*> → adapters → filtered.xsl
  - Delete this file
- <*your project name*> → adapters → <*your adapter name*>.xml
  - Add procedure *addNewCar*, *changeCar*, *deleteCar*, *getInfo*
  - Delete procedures *getStories* and *getStoriesFiltered*

  - Change ‘Connectivity’: add your z/VSE host and port

- <*your project name*> → adapters → <*your adapter name*>-impl.xml
  - Delete functions *getStories*, *getStoriesFiltered*, *getPath*
  - Add code for defined procedures *addNewCar*, *changeCar*, *deleteCar*, *getInfo*. Full source can be found in [5].

```
function getInfo() {
	var ccInstance = new com.ibm.zvse.adapter.skvssampJava();

	return {
		result: ccInstance.getInfo( )
	};
}
```

As you can see in picture above, you need to define an instance of the class to call your java function:

```
var ccInstance = new com.ibm.zvse.adapter.skvssampJava();
```

where *com.ibm.zvse.adapter* – package, *skvssampJava* – class to call

In the next lines function *getInfo*() is called and return result.

Testing and debugging

By this stage of developing you have:

- MobileFirst Adapter
- MobileFirst Java code for Server

To test your adapter, choose “Run As” -> “Invoke MobileFirst Procedure”.

Then choose procedure you would like to test, provide it with parameters, if required, and press “Run”.

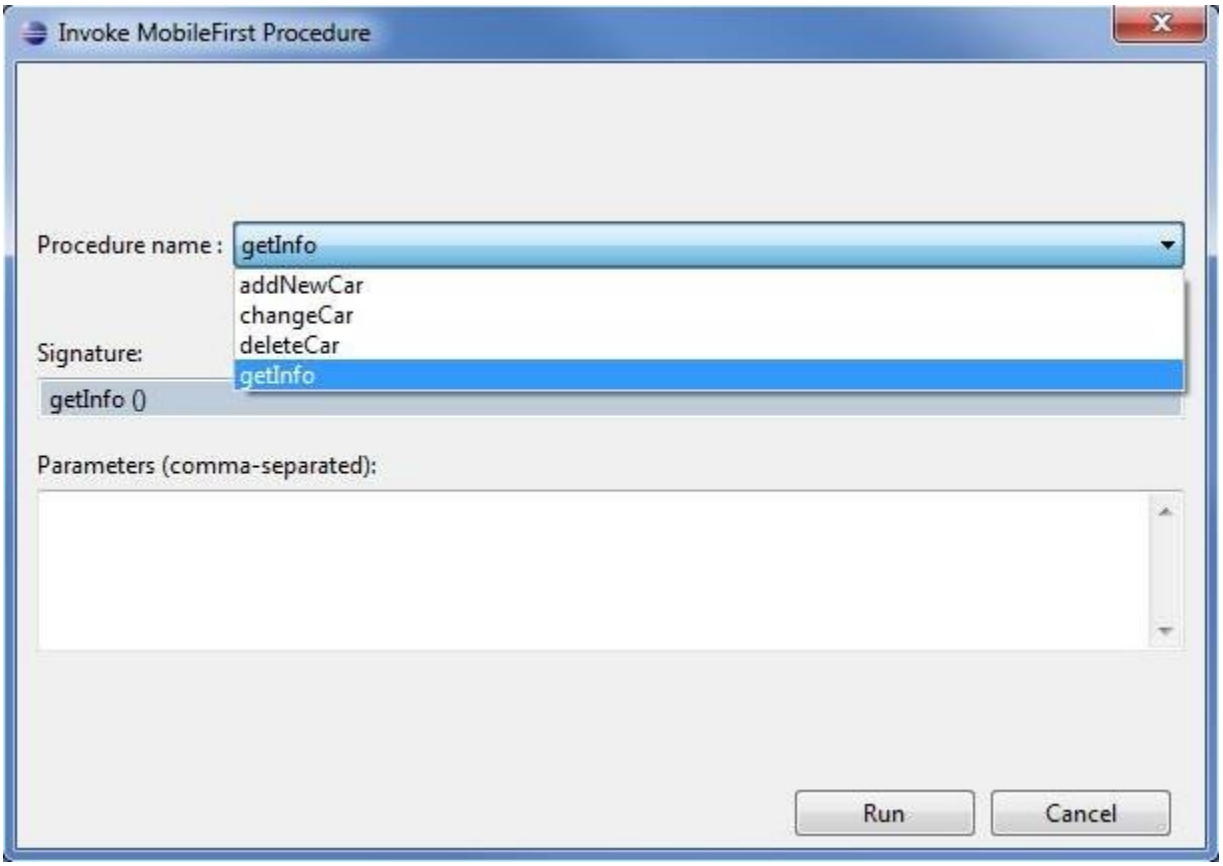

Figure 15. Invocation of MobileFirst procedure

The result will be shown in your browser. For example:

Invocation Result of procedure: 'getInfo' from the MobileFirst Server:


```
{
   "isSuccessful": true,
   "result": "{\"list\":[{\"ARTICLENO\":\"1\",\"MANUFACTURER\":\"Volkswagen          \",\"TYPE\":\"New Beetle          \",\"MODEL
\":\"Petro Model          \",\"HP\":\"115\",\"DISPLACEMENT\":\"2000\",\"CYLINDERS\":\"4\",\"COLOUR\":\"Red
\",\"FEATURES\":\"Sliding Roof        \",\"PRICE\":\"17000\"},{\"ARTICLENO\":\"2\",\"MANUFACTURER\":\"Mustang             \",\"TYPE
\":\"GT 2                 \",\"MODEL\":\"DR CONV             \",\"HP\":\"250\",\"DISPLACEMENT\":\"4600\",\"CYLINDERS\":\"8\",\"COLOUR
\":\"Black                \",\"FEATURES\":\"Smoker's Package     \",\"PRICE\":\"30190\"},{\"ARTICLENO\":\"3\",\"MANUFACTURER
\":\"Ford                 \",\"TYPE\":\"Taurus             \",\"MODEL\":\"SE Station Wagon     \",\"HP\":\"200\",\"DISPLACEMENT
\":\"3000\",\"CYLINDERS\":\"6\",\"COLOUR\":\"Blue                \",\"FEATURES\":\"Appearance Package  \",\"PRICE\":\"23280
\"},{\"ARTICLENO\":\"4\",\"MANUFACTURER\":\"BMW                \",\"TYPE\":\"compact             \",\"MODEL\":\"316i
\",\"HP\":\"102\",\"DISPLACEMENT\":\"1600\",\"CYLINDERS\":\"4\",\"COLOUR\":\"VelvetBlue Metallic \",\"FEATURES\":\"sport
edition       \",\"PRICE\":\"20500\"},{\"ARTICLENO\":\"5\",\"MANUFACTURER\":\"Mercedes            \",\"TYPE\":\"E220
\",\"MODEL\":\"Avantgarde          \",\"HP\":\"160\",\"DISPLACEMENT\":\"2300\",\"CYLINDERS\":\"6\",\"COLOUR\":\"Grey
\",\"FEATURES\":\"Navigation System   \",\"PRICE\":\"67000\"},{\"ARTICLENO\":\"6\",\"MANUFACTURER\":\"Porsche             \",\"TYPE
\":\"Roadster             \",\"MODEL\":\"                    \",\"HP\":\"220\",\"DISPLACEMENT\":\"2700\",\"CYLINDERS\":\"6\",\"COLOUR
\":\"Silver               \",\"FEATURES\":\"Leather,CD Changer  \",\"PRICE\":\"42300\"},{\"ARTICLENO\":\"7\",\"MANUFACTURER
\":\"Ford                 \",\"TYPE\":\"Escort               \",\"MODEL\":\"ZX2 2 Door Coupe     \",\"HP\":\"150\",\"DISPLACEMENT
\":\"2000\",\"CYLINDERS\":\"6\",\"COLOUR\":\"White                \",\"FEATURES\":\"Sp.Seats,Zetec Eng. \",\"PRICE\":\"15715
\"},{\"ARTICLENO\":\"8\",\"MANUFACTURER\":\"Volvo                \",\"TYPE\":\"C30                \",\"MODEL\":\"vcgh
\",\"HP\":\"125\",\"DISPLACEMENT\":\"1798\",\"CYLINDERS\":\"3\",\"COLOUR\":\"grey               \",\"FEATURES
\":\"sport                \",\"PRICE\":\"10000\"},{\"ARTICLENO\":\"9\",\"MANUFACTURER\":\"Volvo                \",\"TYPE
\":\"S60                  \",\"MODEL\":\"sdfsdf              \",\"HP\":\"123\",\"DISPLACEMENT\":\"1\",\"CYLINDERS\":\"2\",\"COLOUR
\":\"blau                 \",\"FEATURES\":\"nice               \",\"PRICE\":\"5000\"}]}"
}
```


Figure 16. Result from z/VSE and MobileFirst Server

As you see in the Figure 16, the result is packed in JSON format. JavaScript you use in your logic (main.js) can work with JSON format, you need just add couple of statements. See function *displayFeeds*() in the main.js code (found in [5]).

Mobile application

Mobile application consists of at least two parts: user interface (described in index.html) and business logic (described in main.js).

For the *usedCar* application, we will create a simple interface: an application header and two buttons: "*Get Info*" to get a list of cars and "*Add a new car*" to add a new record to the cars list. For the section "*add new car*", it is also necessary to define fields that describe new car. Please see the code in [5].

Logic for mobile part is located in main.js and contains information the following data:

- adapter and functions to call
- pack and unpack parameters if necessary
- what to do with the results

Full code can be found in [5].

To test and debug your Mobile application, run it as a "Preview" and use your browser Web development Tools. For example,

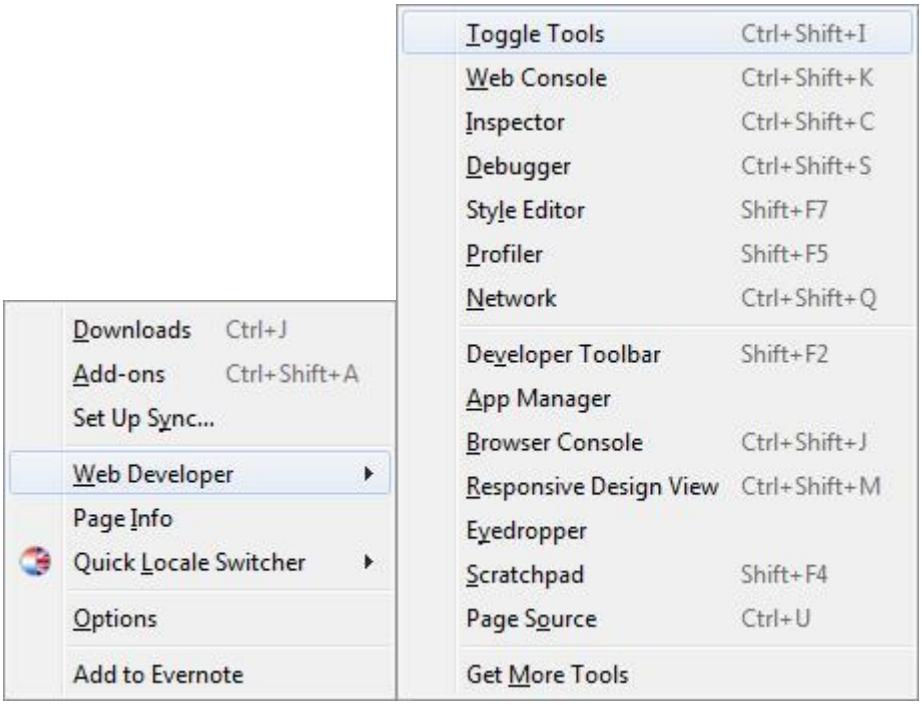

Figure 17 Debugger in browser

As a result of your development, you should get the following screen:

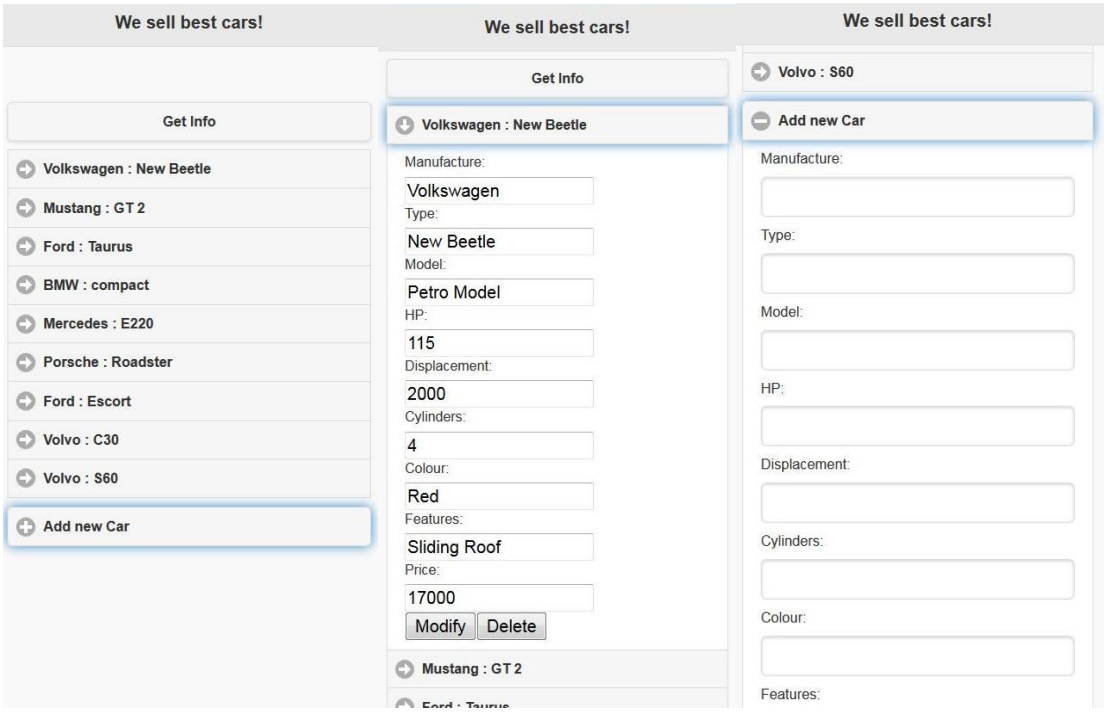

Figure 18. Result of application's work

Adding a MobileFirst environment

All samples above used a common (by default) environment. You could see the mobile application running in your browser. In addition to this environment, you can add a specific MobileFirst environment for any mobile device at any time. To do this, choose *<your project name>* → New → MobileFirst Environment.

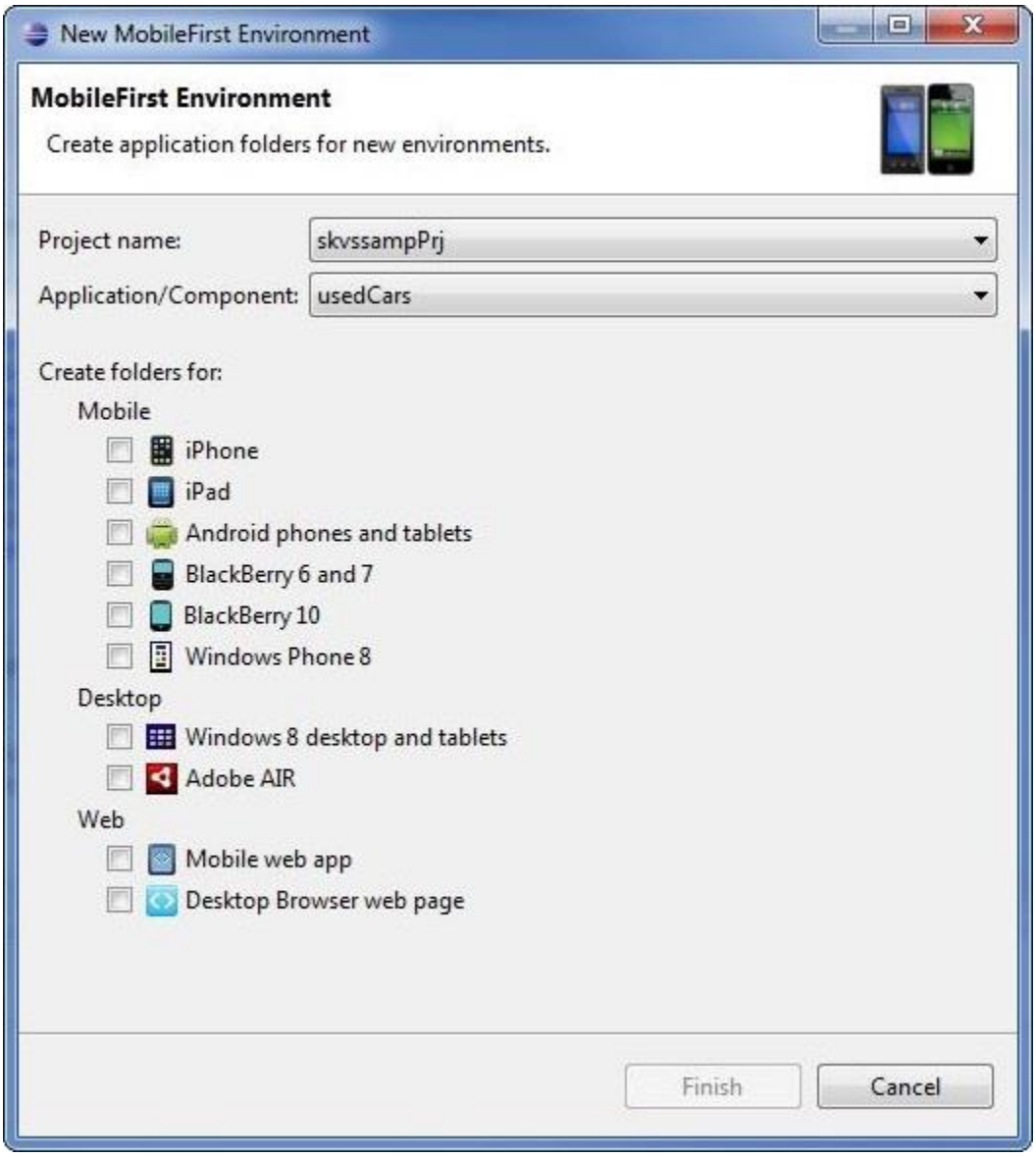

Figure 19. Add an environment

After that, additional folders named after device you include in your project will appear in your project structure.

Now you can test your application running on one of the chosen devices (in browser). If you would like to use more specific functions for your mobile device, please refer to [9] for more information about native development.

Figure 20. Different MobileFirst Environments

Resources

[1] http://www-03.ibm.com/software/products/en/mobilefirstfoundation

[2] https://developer.ibm.com/mobilefirstplatform/

[3] http://www-03.ibm.com/systems/z/os/zvse/products/connectors.html#conn

[4] How to use Web Services with z/VSE
ftp://public.dhe.ibm.com/eserver/zseries/zos/vse/pdf3/HowToUseWebServicesWithzVSE.pdf

[5] Samples source code
ftp://public.dhe.ibm.com/eserver/zseries/zos/vse/download/skvssampPrj.zip

[6] IBM white paper: Native, web or hybrid mobile-app development

ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf

[7] IBM white paper: An overview of IBM MobileFirst Platform

http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14181usen/WSW14181USEN.PDF

[8] User interface design for the mobile web

http://www.ibm.com/developerworks/web/library/wa-interface/index.html

[9] https://developer.ibm.com/mobilefirstplatform/documentation/getting-started/

[10] https://developer.ibm.com/mobilefirstplatform/documentation/getting-started-hybrid-development-6-3/

[11] CICS2WS Toolkit http://www-03.ibm.com/systems/z/os/zvse/downloads/#cics2ws

[12] z/VSE Connector Client http://www-03.ibm.com/systems/z/os/zvse/downloads/#vsecon

[13] jQuery Mobile http://jquerymobile.com

Disclaimer

This publication is intended to help VSE system programmers to set up infrastructure for their operating environments. The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk. Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment. Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of IBM®:

IBM®

Worklight®

IBM MobileFirst™

z/VSE®

BlueMix™

For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows XP, .Net, .Net logo, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.