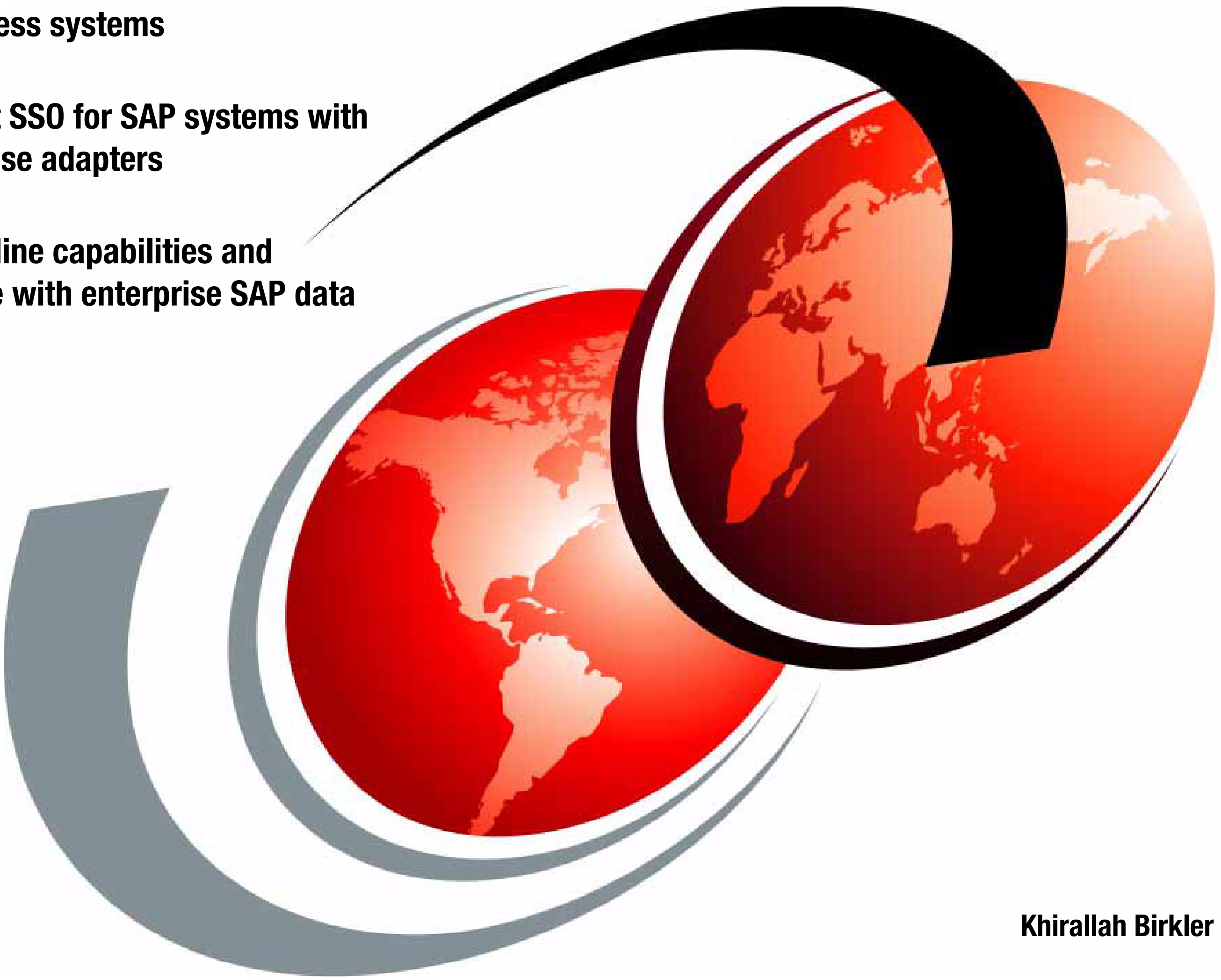
Extending SAP Solutions to the Mobile Enterprise with IBM MobileFirst Platform Foundation

Learn how to enable mobile access to SAP business systems

Implement SSO for SAP systems with ready-to-use adapters

Exploit offline capabilities and sychronize with enterprise SAP data



Khirallah Birkler

Redpaper

ibm.com/redbooks

International Technical Support Organization

Extending SAP Solutions to the Mobile Enterprise with IBM MobileFirst Platform Foundation

February 2015

REDP-5136-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (February 2015)

This edition applies to Version 6, Release 3, Modification 0 of IBM MobileFirst Platform Foundation (product number 5725-I43).

This document was created or updated on February 20, 2015.

Contents

Notices

Preface

SAP is a market leader in enterprise business application software. SAP solutions provide a rich set of composable application modules, and configurable functional capabilities that are expected from a comprehensive enterprise business application software suite.

Enabling mobile access to SAP business functions and data for enterprise clients, employees, and business partners is a typical requirement for SAP projects.

This IBM® Redpaper™ publication describes the benefits of the IBM MobileFirst™ Platform Foundation security framework, which is an essential building block of the IBM MobileFirst platform. This paper also discusses key capabilities in the IBM MobileFirst Platform to authenticate with SAP business systems from mobile applications.

The scenarios in this paper demonstrate these features:

- How to develop a custom login module with IBM MobileFirst to authenticate mobile user access to an SAP enterprise resource planning (ERP) system.
- How to implement single sign-on (SSO) authentication in a mobile application to access SAP business systems using a pre-built adapter for IBM Cast Iron® included with IBM MobileFirst and the ready-to-use adapter for SAP NetWeaver Gateway, also included with IBM MobileFirst.
- How to take advantage of the offline capabilities included in IBM MobileFirst Platform Foundation V6.3 to store business data locally, act on this data, and synchronize the data with the originating SAP ERP system.

The sample code delivered with this paper can be modified and used in projects that require the integration of mobile apps that are developed with IBM MobileFirst Platform Foundation and business systems that run in the SAP domain.

This paper is for mobile application developers and technical consultants who design and build systems of engagement to interact with SAP solutions in the heterogeneous enterprise.

Readers of this paper can also benefit from the IBM Redbooks® publication *IBM Software for SAP Solutions*, SG24-8230.

Authors

This paper was produced by Khirallah Birkler, IBM and SAP Mobile Solution Architect in the IBM Software Services for WebSphere®, Mobile Practice, working in partnership with the International Technical Support Organization (ITSO).

Khirallah Birkler is a Certified IT Specialist and IBM/SAP Mobile Solution Architect in the IBM SWG division, IBM Germany. As an Architect, Khirallah helps clients to identify requirements and design mobile solutions based on the IBM MobileFirst portfolio. A special focus of his work is the integration of enterprise data into mobile scenarios, which is always a key requirement for enterprise customers. Khirallah holds a Bachelor of Science degree in Information Technology Management from the University of Cooperative Education, Stuttgart.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks Project Leader with ITSO, Global Content Services.

Thanks to the following people for their contributions to this project:

Uwe Klein
IBM Global Business Services®, SAP NetWeaver Integration and Mobility, IBM Germany

Peter Bahrs
IBM MobileFirst Platform Foundation CTO, AIM Lab Services, IBM Software Group

Karl Bishop
IBM MobileFirst Platform Product Management, IBM Software Group

Christopher Jaun
IBM MobileFirst Mobile Web Tools development, IBM Software Group

Artem Spector
IBM MobileFirst Platform Foundation Architect, IBM Software Group

Greg Truty
IBM MobileFirst Platform Chief Architect, IBM Software Group

Now you can become a published author, too!

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Mobile authentication with SAP Business Suite

This chapter describes the benefits of the IBM MobileFirst Platform Foundation security framework, which is an essential building block of the platform.

This chapter demonstrates how to develop a custom login module within the MobileFirst security framework to authenticate mobile user access to an SAP enterprise resource planning (ERP) system.

This chapter includes sample code that can be modified and used in projects that require the integration of mobile apps that are developed with the IBM MobileFirst Platform and business system that run in the SAP domain.

1.1 IBM MobileFirst Platform Foundation security framework

The IBM MobileFirst Platform Foundation security framework serves two main goals:

- Controls access to the protected resources.
- Propagates the user (or server) identity to the back-end systems through the adapter framework.

Key to the success of the mobile application is that the MobileFirst security framework does not include its own user registry, credentials storage, or access control management. Instead, it delegates those functions to the existing enterprise security infrastructure. This delegation allows the MobileFirst Server to integrate smoothly as a presentation tier into the existing enterprise landscape. Integration with the existing security infrastructure is an important feature of the MobileFirst platform security framework. The security framework supports custom extensions that allow integration with virtually any security mechanism within the SAP security domain.

For common mechanisms, ready-to-use login modules are included with the product, such as LDAP authentication, user ID and password credentials, and client certificates. Besides these assets provided with the product, developers can define a custom login module by using a well-defined API to leverage any external party as authentication and authorization master.

The IBM MobileFirst platform security framework consists of the key components shown in Figure 1-1.

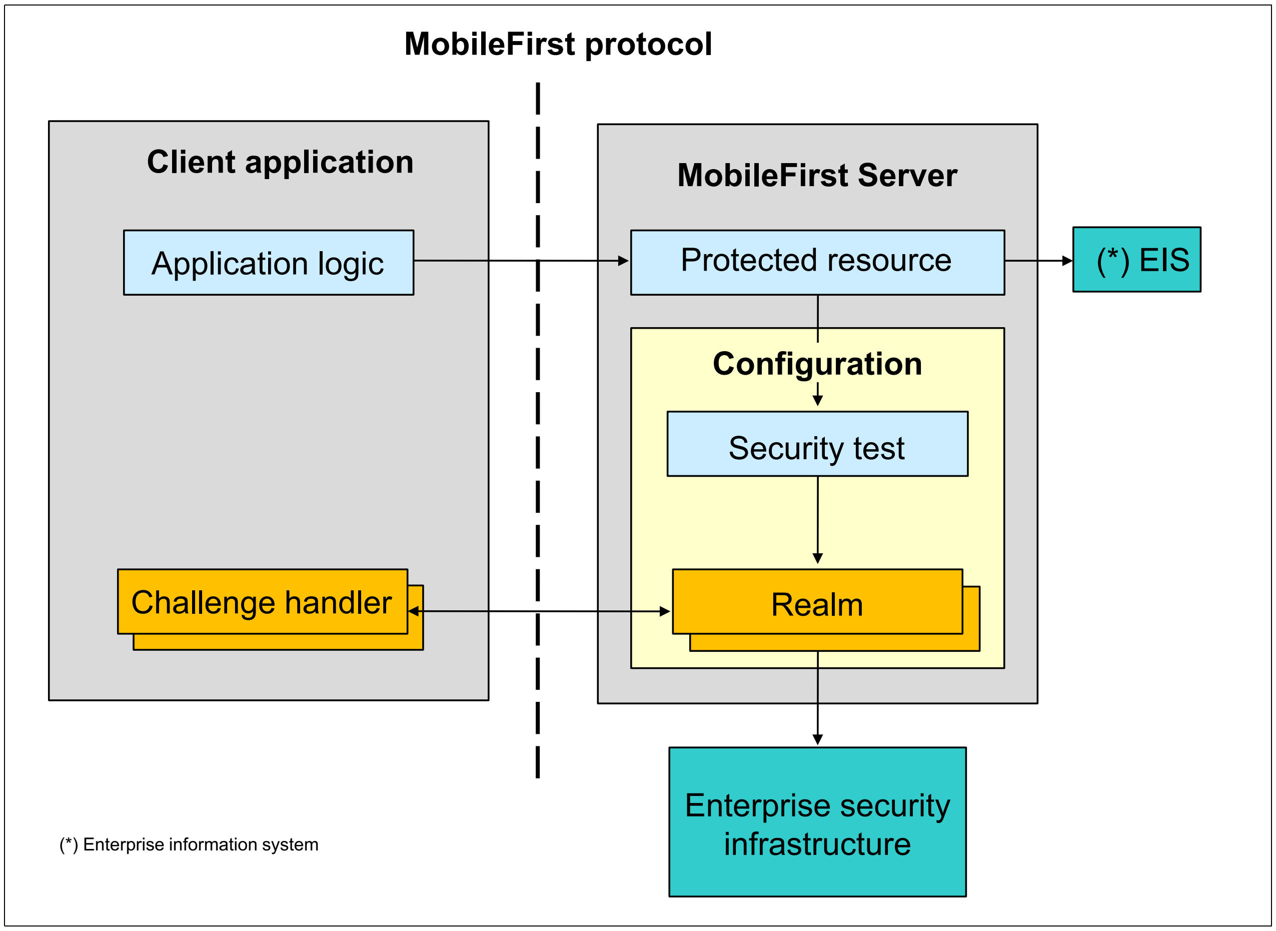


Figure 1-1 MobileFirst security framework

1.1.1 Protected resources and authentication context

The MobileFirst Server controls the security sessions and access to protected resources. A protected resource can be any of the following items:

- Application

  Requests to the application require successful authentication in all realms of the security test that is defined in the application descriptor.

- Adapter procedure

  Procedure invocation requires successful authentication in all realms of the security test that is defined in the adapter descriptor. The user identity and credentials that are obtained during such authentication can be propagated to the enterprise information system represented by this adapter.

- Event source

  Subscription to push notifications requires successful authentication in all realms of the security test that are defined in the event source definition (in adapter JavaScript).

- Static resource

  Static resources are defined as URL patterns in the authentication configuration file. They allow protection of *static* web applications such as the MobileFirst Console.

During the session, an application can access different resources. The results of the authentication in different realms are stored in the session authentication context. These results are then shared among all of the protected resources in the scope of the current session.

1.1.2 Realms and security tests

A realm represents a fully configured security check that must be completed before it can allow access to a protected resource. The semantics of the checks are not limited to the authentication but can implement any logic that can serve as protection for the server-side application resources, as in these examples:

- User authentication
- Device authentication and provisioning
- Application authenticity check
- Remote disable of the ability to connect to the MobileFirst Server
- Direct update
- Anti-XSRF (cross-site request forgery) check

The *realms* are defined in the authentication configuration file in the MobileFirst project. A realm consists of two parts: the authenticator and the login module. The authenticator obtains the credentials from the client, and the login module validates those credentials, and builds the user identity.

The realms are grouped into *security tests*, which are defined in the same authentication configuration files. The security test defines the group of realms, and also the order in which they must be checked. For example, an approach that often makes sense is not to ask for the user credentials before verifying that the application itself is authentic. Also possible is to create a custom security test from scratch.

1.1.3 Authenticators and login modules

An authenticator is a server component that is used to collect credentials from the client. The authenticator passes the credentials to a login module, which validates them and builds a client identity object. Both authenticators and login modules are components of the application's realm. Several predefined authenticators and login modules for common scenarios are supplied with the product and developers can write custom authenticators and login modules for specific project requirements.

1.1.4 MobileFirst security protocol and client challenge handlers

Each security check defines its own protocol, which is a sequence of challenges that are sent by the server and responses that are sent by the client. On the server side, the component that implements this private protocol is the *authenticator*. On the client side, the corresponding component is called the *challenge handler*.

When the client request tries to access a protected resource, the MobileFirst Server checks all the appropriate realms. Several realms can decide to send a challenge. Challenges from multiple realms are composed into a single response and sent back to the client.

The IBM MobileFirst client-side infrastructure extracts the individual challenges from the response, and routes them to the appropriate challenge handlers. When a challenge handler finishes the processing, it submits its response to the MobileFirst client-side infrastructure. The MobileFirst Server extracts those responses from the request and passes them to the appropriate authenticators.

If an authenticator is satisfied, it reports a success, and the MobileFirst Server calls the login module. If the login module succeeds in validating all of the credentials, the realm is considered successfully authenticated. If all the realms of the security test are successfully authenticated, the MobileFirst Server allows the request processing to proceed.

If a realm check fails, its authenticator sends another (or the same) challenge to the client, and the whole process repeats.

During the session, an application can access different resources. The results of the authentication in different realms are stored in the session authentication context. These results are then shared among all protected resources in the scope of the current session.

1.2 Developing mobile authentication logic for SAP applications

This section describes the steps to develop a custom login module to authenticate with SAP Business Suite applications. This section also covers the creation of a basic mobile application that is used to demonstrate the SAP integration capabilities.

1.2.1 Create a hybrid application skeleton by using MobileFirst Studio

Complete the following steps to create a new hybrid application by using the corresponding Eclipse wizard in the MobileFirst Studio:

1. Select **MobileFirst Project** (Figure 1-2).

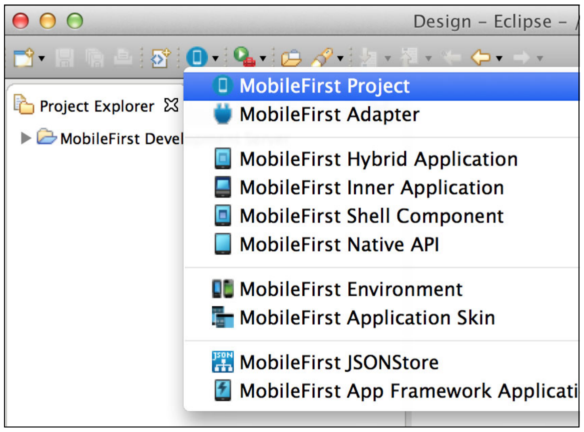
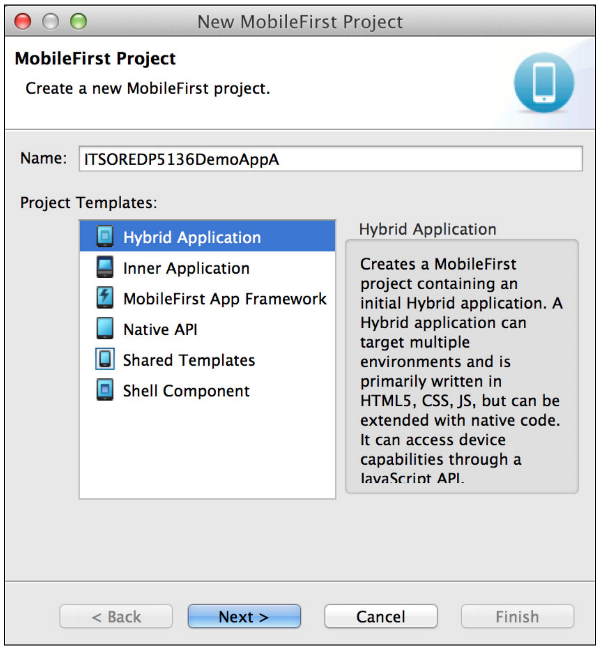


Figure 1-2 Create a new MobileFirst project

2. Enter a project name in the Name field, for example, `ITSOREDP5136DemoAppA`, and select project template **Hybrid Application** (Figure 1-3).



New MobileFirst Project

MobileFirst Project

Create a new MobileFirst project.

Name: ITSOREDP5136DemoAppA

Project Templates:

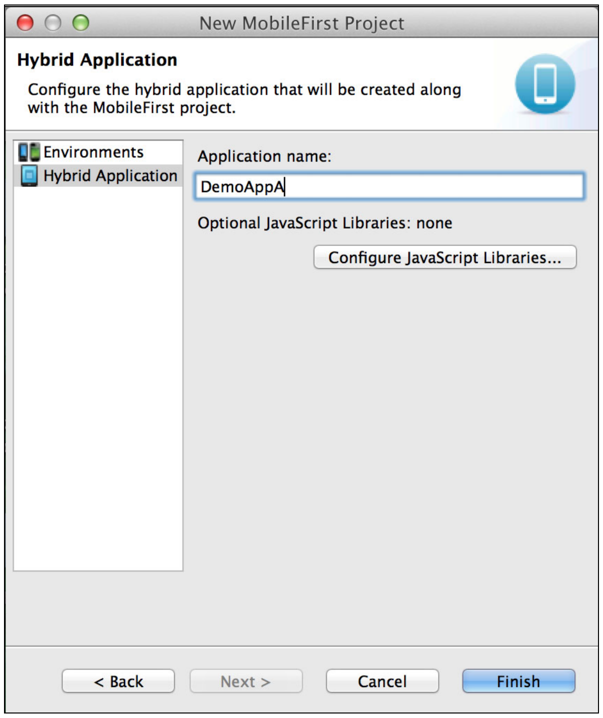
- Hybrid Application
- Inner Application
- MobileFirst App Framework
- Native API
- Shared Templates
- Shell Component

Hybrid Application

Creates a MobileFirst project containing an initial Hybrid application. A Hybrid application can target multiple environments and is primarily written in HTML5, CSS, JS, but can be extended with native code. It can access device capabilities through a JavaScript API.

Figure 1-3 Enter the project name and select the project template type

3. Enter a meaningful application name, for example, `DemoAppA`, and click **Finish** (Figure 1-4).



Configure the hybrid application that will be created along with the MobileFirst project.

Application name: DemoAppA

Optional JavaScript Libraries: none

Figure 1-4 Enter the mobile application name

At this point you have created an empty hybrid application project. It is extended step-by-step in the following sections, demonstrating how to create a MobileFirst custom login module, which enables authentication with SAP ERP.

1.2.2 Create a new MobileFirst adapter in the hybrid application project

To create a MobileFirst adapter, complete the following steps:

1. Click the **Create a MobileFirst Artifact** icon and select **MobileFirst Adapter** (Figure 1-5).

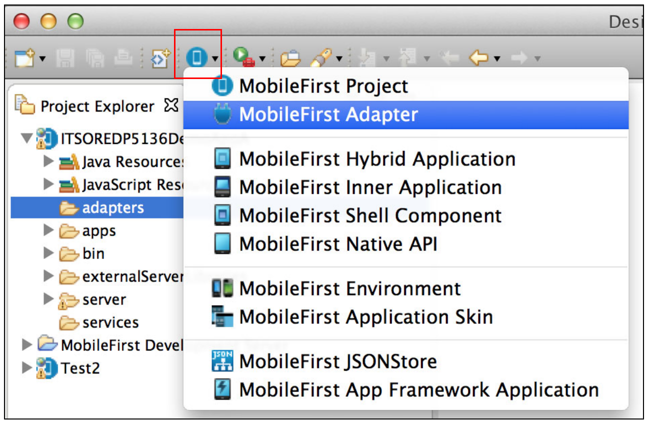
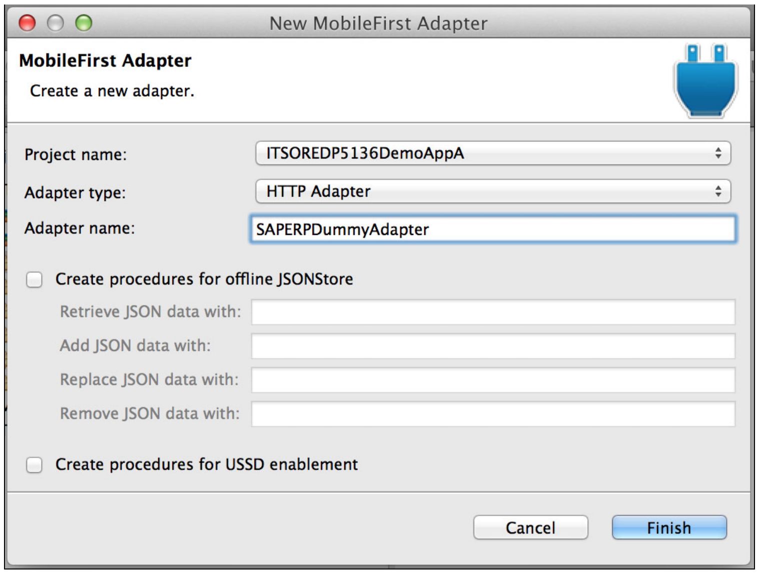


Figure 1-5 Create a new MobileFirst Adapter

2. Select adapter type **HTTP Adapter** and enter a meaningful adapter name, for example `SAPERPDummyAdapter` (Figure 1-6), and then click **Finish**.



Project name: ITSOREDP5136DemoAppA
Adapter type: HTTP Adapter
Adapter name: SAPERPDummyAdapter

☐ Create procedures for offline JSONStore
Retrieve JSON data with:
Add JSON data with:
Replace JSON data with:
Remove JSON data with:

☐ Create procedures for USSD enablement

Figure 1-6 Enter the adapter name and select the adapter type

1.2.3 Modify the adapter XML file

The adapter XML file defines technical details of the generated MobileFirst HTTP adapter. In this scenario, the adapter is used as a dummy adapter to represent a resource that can be secured and returns business data. An important aspect is the definition of the `securityTest` attribute, which forces the IBM MobileFirst Platform security framework to run this procedure only when a valid security context is available. To modify the adapter XML file for scenario, complete the following steps:

1. Open the XML file `SAPERPDummyAdapter.xml`.
2. Add a procedure for the operation `getSecreteBusinessData` (Example 1-1).

Example 1-1 Declaration of procedure getSecretBusinessData in SAPERPDummyAdapter.xml

```
<procedure name="getSecretBusinessData" securityTest="SAPNetWeaver-securityTest"/>
```

3. Save the adapter XML file.

1.2.4 Modify the adapter JavaScript file

The adapter JavaScript file contains the JavaScript logic that represents the IBM MobileFirst adapter. In this scenario, it contains one function that returns a static string with the name `secretBusinessData`. Complete the following steps:

1. Open the JavaScript file `SAPERPDummyAdapter-impl.js`.
2. Add a JavaScript function, which is shown in Example 1-2 on page 8.

Example 1-2 JavaScript function getSecretBusinessData in SAPERPDummyAdapter-impl.js

```
function getSecretBusinessData(input) {
return   {
   secretBusinessData: 'ITSO Redbooks - REDP5136'
         };
}
```

3. Save the adapter JavaScript file.

1.2.5 Modify the authenticationConfig.xml file

The `authenticationConfig.xml` file controls the security behavior of the mobile app and its subcomponents. In this scenario, a MobileFirst adapter procedure is secured and accessible only when a specific security context is available. The following three elements must be created:

- A `customSecurityTest` element
- A realm element
- A login module element

The `customSecurityTest` element points to a valid realm element. The realm element defines a suitable authenticator class and points to a valid login module element. The login module element defines the SAP system used for authentication.

To modify the security XML file in this scenario, complete the following steps:

1. Open the `authenticationConfig.xml` file, which is in the `server/conf` directory, and select the **Source** view.
2. Add an XML section for a `customSecurityTest`, which is shown in Example 1-3.

Example 1-3 Add customSecurityTest section

```
<customSecurityTest name="SAPNetWeaver-securityTest">
<test isInternalDeviceID="false" isInternalUserID="true"
realm="SAPNetWeaverRealm"/>
</customSecurityTest>
```

3. Add an XML section for a `realm` (Example 1-4).

Example 1-4 Add realm section

```
<realm loginModule="SAPNetWeaverLoginModule" name="SAPNetWeaverRealm">

<className>com.ibm.CustomWorkLightAuthenticators.SAPNetWeaverAuthenticator</cla
ssName>
</realm>
```

4. Add an XML section for a `loginModule`, which is shown in Example 1-5.

Example 1-5 Add loginModule section

```
      <loginModule name="SAPNetWeaverLoginModule">

<className>com.ibm.CustomWorkLightLoginModules.SAPNetWeaverLoginModule</className>
             <parameter name="sapsystemtype" value="${sap.SAPSYSTEMTYPE}"/>
             <parameter name="hostname" value="${sap.SAPHOSTNAME}"/>
             <parameter name="sysnr" value="${sap.SAPSYSNR}"/>
             <parameter name="client" value="${sap.SAPCLIENT}"/>
             <parameter name="sid" value="${sap.SAPSID}"/>
      </loginModule>
```

5. Save the `authenticationConfig.xml` file.

1.2.6 Modify the worklight.properties file

The `worklight.properties` file controls key parameters of the MobileFirst Server. In this scenario, several environment parameters are defined, which are read by the custom login module when it is instantiated. A good practice is to define environment-specific parameters in this file to make the deployment architecture more flexible.

The custom login module implemented in this scenario requires five parameters to be defined in the `worklight.properties` file. To add the five parameters to the `worklight.properties` file, complete the following steps:

1. Open the `worklight.properties` file.
2. Add the section that is shown in Example 1-6.

Example 1-6 Parameters defining the location of the SAP ABAP ERP system

```
#################################################################################
#  IBM SAP NetWeaver Login module specific parameters
#################################################################################
# For SAP ABAP system login module
sap.SAPSYSTEMTYPE=ABAP
sap.SAPHOSTNAME=saphostname
sap.SAPSYSNR=sapsysnr
sap.SAPCLIENT=sapclientno
sap.SAPSID=sapsid
```

3. Replace all Italic values in that section with the data of your SAP system.
4. Save the `worklight.properties` file.

1.2.7 Implement a client-side JavaScript security challenge handler

The client-side JavaScript challenge handler is a key component of the IBM MobileFirst Platform security framework. The challenge handler is located in the hybrid application client-side and interacts with the custom authentication.

In general, a security challenge handler is a client-side JavaScript file that implements various functions. The server-side security framework expects to interact with these functions in a predefined way.

Complete the following steps to implement a client-side JavaScript security challenge handler:

1. Create a new file in the `common/js` folder and name it `SAPNetWeaverRealmChallengeHandler.js`.
2. Add the code shown in Example 1-7 to initialize the challenge handler.

Example 1-7 Security challenge handler initialization

```
/// ////////////////////
// Challenge Handler
// ////////////////////
if (!window.$) { window.$ = WLJQ;}

var sapNetWeaverRealmChallengeHandler =
WL.Client.createChallengeHandler("SAPNetWeaverRealm");
```

3. Add the code for the `isCustomResponse` function, as shown in Example 1-8.

Example 1-8 Security challenge handler isCustomResponse function

```
sapNetWeaverRealmChallengeHandler.isCustomResponse = function(response) {

    if (!response || !response.responseJSON) {
    console.log("SAPNetWeaverRealmChallengeHandler.isCustomResponse: no
response object - no responseJSON object");
        return false;
    }

    if (response.responseJSON.authStatus) {
    console.log("SAPNetWeaverRealmChallengeHandler.isCustomResponse: authStatus
= " + response.responseJSON.authStatus);
    return true;
    }
    else
    {
    console.log("SAPNetWeaverRealmChallengeHandler.isCustomResponse: no
authStatus object");
    return false;
    }
};
```

4. Add the code for the `handleChallenge` function, as shown in Example 1-9.

Example 1-9 Security challenge handler handleChallenge function

```
sapNetWeaverRealmChallengeHandler.handleChallenge = function(response){
   var authStatus = response.responseJSON.authStatus;

   if (authStatus === "required"){
      console.log("SAPNetWeaverRealmChallengeHandler.handleChallenge:
required");

      if(response.responseJSON.errorMessage){
       console.error(response.responseJSON.errorMessage);
       // errors occured during login - just pass error message and clear
password field
       $('#loginError').html('Login error - UserID or Password invalid.');
```

```
        $('#passwordInputField').val('');
        }else
        {
        // switch view to hide display view and show login view
        $('#AppLoginView').show();
          $('#AppHomeView').hide();

      }

   } else if (authStatus === "complete"){
      console.log("SAPNetWeaverRealmChallengeHandler.handleChallenge:
complete");

      // switch view to hide login view and display main view
      $('#AppHomeView').show();
      $('#AppLoginView').hide();

      // send success notification to handler
      sapNetWeaverRealmChallengeHandler.submitSuccess();

   }
};
```

5. Add the code for the `submitLoginFormCallback` function that is shown in Example 1-10.

Example 1-10 Security challenge handler submitLoginFormCallback function

```
sapNetWeaverRealmChallengeHandler.submitLoginFormCallback = function(response)
{
   console.log("SAPNetWeaverRealmChallengeHandler.submitLoginFormCallback");
   var isLoginFormResponse =
sapNetWeaverRealmChallengeHandler.isCustomResponse(response);

    if (isLoginFormResponse){
    sapNetWeaverRealmChallengeHandler.handleChallenge(response);
    }
};
```

6. Save the `SAPNetWeaverRealmChallengeHandler.js` file.

1.2.8 Extend the main.js JavaScript file

The `main.js` JavaScript file contains application-specific code. In this scenario, the file contains the JavaScript function to call the MobileFirst adapter procedure. In addition, it contains the functions that are called when the various buttons are pressed.

1. Open the `main.js` file.
2. Add the code snippet, shown in Example 1-11 on page 12, to call the adapter procedure that returns the static business data from the back-end system.

Example 1-11 Calling the adapter procedure to receive back-end data

```
function getSecretBusinessData(){
   var invocationData = {
         adapter: "SAPERPDummyAdapter",
         procedure: "getSecretBusinessData",
         parameters: []
   };

   WL.Client.invokeProcedure(invocationData, {
      onSuccess: getSecretBusinessData_Callback,
      onFailure: getSecretBusinessData_Callback
   });
}
```

3. Add the code snippet, shown in Example 1-12, to process the received business data.

Example 1-12 Implementing the callback function to process results

```
function getSecretBusinessData_Callback(response){
   if(response.invocationResult.isSuccessful) {

      document.getElementById('AppHomeResults').innerHTML = "Secret business
data is: " + JSON.stringify(response.invocationResult.secretBusinessData);

      if(response && (response.invocationResult) &&
(response.invocationResult['WL-Authentication-Success']) &&
(response.invocationResult['WL-Authentication-Success']['SAPNetWeaverRealm']))
      {
         var realm =
response.invocationResult['WL-Authentication-Success']['SAPNetWeaverRealm'];
         document.getElementById('AppHomeLoginDetails').innerHTML = "Logged in
as User: " + realm.userId;

      }
   }else
   {
         document.getElementById('AppHomeResults').innerHTML =
JSON.stringify(response);
      }
}
```

4. Add the code snippet, shown in Example 1-13, to define the actions that are triggered when the corresponding buttons are clicked.

Example 1-13 Adding logic to handle button interactions

```
$('#loginButton').bind('click', function () {
    var reqURL = '/sapnetweaversecuritycheck';
    var options = {};
    options.parameters = {
        username : $('#usernameInputField').val(),
        password : $('#passwordInputField').val()
    };
    options.headers = {};
    sapNetWeaverRealmChallengeHandler.submitLoginForm(reqURL, options,
sapNetWeaverRealmChallengeHandler.submitLoginFormCallback);
});

$('#cancelButton').bind('click', function () {
   $('#AppHomeView').show();
   $('#AppLoginView').hide();
   sapNetWeaverRealmChallengeHandler.submitFailure();
});
```

5. Save the `main.js` file.

1.2.9 Implement the mobile application entry page

The user interface of the generated mobile hybrid application is included in the `index.html` file.

In this scenario, a screen displays two buttons. One button calls a secured business function and the other button forces the logout mechanism of the IBM MobileFirst Platform security framework. The security framework ensures that a valid security context exists before the secured function runs. When no security context is discovered, the application displays a login page where the user can provide user ID and password credentials. These credentials are verified by the custom login module logic created in the previous sections.

Example 1-14 shows the `index.html` page that contains the mobile application logic.

Example 1-14 Implementation of the index.html page

```
<!DOCTYPE HTML>
<html>
    <head>
      <meta charset="UTF-8">
      <title>Login Module Test</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0, user-scalable=0">
      <link rel="shortcut icon" href="images/favicon.png">
      <link rel="apple-touch-icon" href="images/apple-touch-icon.png">
      <link rel="stylesheet" href="css/main.css">
      <script>window.$ = window.jQuery = WLJQ;</script>
    </head>
    <body style="display: none;">
      <div id="header" style="color: black; font-family: Arial">
            <h1>SAP NetWeaver Custom Login Module</h1>
```

```
        </div>

        <div id="wrapper">
        <div id="AppHomeView">
              <div class="wrapper">
                 <input type="button" class="appButton" value="Call protected
adapter proc" onclick="getSecretBusinessData()" />
                 <input type="button" class="appButton" value="Logout"
onclick="WL.Client.logout('SAPNetWeaverRealm',{onSuccess: WL.Client.reloadApp})"
/>
              </div>
              <p>
              <div class="wrapper" id="AppHomeResults" style="color: black;
font-size: x-small; font-family: Arial"></div>
              <p>
              <div class="wrapper" id="AppHomeLoginDetails" style="color: blue;
font-size: x-small; font-family: Arial"></div>
           </div>

           <div id="AppLoginView" style="display: none">
              <div id="loginForm"><br />
                 <p id="loginError" style="color: red; font-size: x-small;
font-family: Arial"></p>
                 <input type="text" id="usernameInputField" placeholder="Enter
username" /><br /><br />
                 <input type="password" id="passwordInputField"
placeholder="Enter password"/><br/>
                 <br>
                 <input type="button" id="loginButton" class="formButton"
value="Login" /><input type="button" id="cancelButton" class="formButton"
value="Cancel" />
              </div>
           </div>
        </div>
      <script src="js/initOptions.js"></script>
      <script src="js/main.js"></script>
      <script src="js/messages.js"></script>
      <script src="js/SAPNetWeaverRealmChallengeHandler.js"></script>
    </body>
</html>
```

1.2.10 Implement the server-side Java logic for the custom login module

The custom login logic consists of various artifacts on the server side. An authenticator component is responsible to collect the credentials and the login module code verifies these credentials using the SAP Java Connector (SAP JCo) libraries to connect to the SAP ABAP ERP system.

The SAP connection is done by using a dynamic destination data provider. It enables the IBM MobileFirst platform security framework to interact dynamically with the SAP target system.

The MobileFirst security framework receives an SAP Logon Ticket when the login is successful.

Import the SAP Java Connector library

The SAP Java Connector is available for SAP customers through the SAP Service Marketplace. A valid S-user ID is required to get access to the SAP Java Connector libraries. Read the documentation for your operating system carefully because differences exist for each platform. Complete the following steps:

1. Download the SAP Java Connector 3.0.x libraries for your target operating system from the SAP Service Marketplace:

   `http://service.sap.com/connectors`

   You must have an SAP Service Market place user ID and password to log in.

2. Import the `sapjco3.jar` file to the `server/lib` directory of the MobileFirst project.

Implement the custom authenticator code

The authenticator component implements the `WorklightAuthenticator` interface and overrides the mandatory methods as shown in Example 1-15.

Example 1-15 SAPNetWeaverAuthenticator implementation

```
public class SAPNetWeaverAuthenticator implements WorkLightAuthenticator{
   private static final Logger logger =
Logger.getLogger(SAPNetWeaverAuthenticator.class.getName());
   private static final String LOGIN_PAGE_SECURITY_INDICATOR = "sapnetweaversecuritycheck";

   private Map<String, Object> authenticationData = null;

   @Override
   public void init(Map<String, String> options) throws MissingConfigurationOptionException
{
      logger.info("SAPNetWeaverAuthenticator initialized");
   }

   @Override
   public AuthenticationResult processRequest(HttpServletRequest request,
HttpServletResponse response, boolean isAccessToProtectedResource) throws IOException,
ServletException {
      logger.info("SAPNetWeaverAuthenticator processrequest");
      if (request.getRequestURI().contains(LOGIN_PAGE_SECURITY_INDICATOR)){
         String username = request.getParameter("username");
         String password = request.getParameter("password");
         //logger.info("IBMBluePagesAuthenticator User:" + username + " - Password: " +
password);
         logger.info("SAPNetWeaverAuthenticator User:" + username);

         if (null != username && null != password && username.length() > 0 &&
password.length() > 0){
            authenticationData = new HashMap<String, Object>();
            authenticationData.put("username", username);
            authenticationData.put("password", password);
            return AuthenticationResult.createFrom(AuthenticationStatus.SUCCESS);
         } else {
            response.setContentType("application/json; charset=UTF-8");
            response.setHeader("Cache-Control", "no-cache, must-revalidate");
            response.getWriter().print("{\"authStatus\":\"required\",
\"errorMessage\":\"Please enter username and password\"}");
            return
AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
         }
      }
```

```
        if (!isAccessToProtectedResource)
            return
AuthenticationResult.createFrom(AuthenticationStatus.REQUEST_NOT_RECOGNIZED);

        response.setContentType("application/json; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache, must-revalidate");
        response.getWriter().print("{\"authStatus\":\"required\"}");
        return
AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
    }

    @Override
    public boolean changeResponseOnSuccess(HttpServletRequest request, HttpServletResponse
response) throws IOException {

        logger.info("SAPNetWeaverAuthenticator authentication success - change response");
        if (request.getRequestURI().contains(LOGIN_PAGE_SECURITY_INDICATOR)){
            response.setContentType("application/json; charset=UTF-8");
            response.setHeader("Cache-Control", "no-cache, must-revalidate");
            response.getWriter().print("{\"authStatus\":\"complete\"}");
            return true;
        }
        return false;
    }

    @Override
    public AuthenticationResult processAuthenticationFailure(HttpServletRequest request,
HttpServletResponse response,
            String errorMessage) throws IOException, ServletException {

        logger.info("SAPNetWeaverAuthenticator authentication failed");
        response.setContentType("application/json; charset=UTF-8");
        response.setHeader("Cache-Control", "no-cache, must-revalidate");
        response.getWriter().print("{\"authStatus\":\"required\", \"errorMessage\":\"" +
errorMessage + "\"}");
        return
AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
    }


    @Override
    public AuthenticationResult processRequestAlreadyAuthenticated(HttpServletRequest
request, HttpServletResponse response) throws IOException, ServletException {
        logger.info("SAPNetWeaverAuthenticator already authenticated");
        return AuthenticationResult.createFrom(AuthenticationStatus.REQUEST_NOT_RECOGNIZED);
    }


    @Override
    public Map<String, Object> getAuthenticationData() {
        return authenticationData;
    }

    @Override
    public HttpServletRequest getRequestToProceed(HttpServletRequest request,
HttpServletResponse response, UserIdentity userIdentity)throws IOException {
        return null;
    }
```

```
   @Override
    public WorkLightAuthenticator clone() throws CloneNotSupportedException {
       SAPNetWeaverAuthenticator otherAuthenticator = (SAPNetWeaverAuthenticator)
super.clone();
        otherAuthenticator.authenticationData = new HashMap<String,
Object>(authenticationData);
        return otherAuthenticator;
    }

}
```

The Java class that implements the authenticator must be created in the MobileFirst project, folder `server/java`.

Implement the custom login module code

The custom login module class implements the `WorkLightAuthLoginModule` interface and overrides the mandatory methods, as shown in Example 1-16.

Example 1-16 SAPNetWeaverLoginModule implementation

```
public class SAPNetWeaverLoginModule implements WorkLightAuthLoginModule {

   private static final Logger logger =
Logger.getLogger(SAPNetWeaverLoginModule.class.getName());

   private String USERNAME;
   private String PASSWORD;
   private String SAPTICKET;

   // parameters for SAP ABAP
   private String SAPSYSTEMTYPE;
   private String SAPHOSTNAME;
   private String SAPSYSNR;
   private String SAPCLIENT;
   private String SAPSID;
   private String DESTINATION;

   @Override
   public void init(Map<String, String> options) throws MissingConfigurationOptionException
{
      logger.info("SAPNetWeaverLoginModule init method called.");
      logger.info("options Map: " + (null == options ? "null" : options.toString()));

      if (options.containsKey("sapsystemtype"))
         SAPSYSTEMTYPE = options.get("sapsystemtype");
      else
         throw new MissingConfigurationOptionException("sapsystemtype");

      if(SAPSYSTEMTYPE.equals("ABAP"))
         readABAPConfiguration(options);


        logger.info("SAPNetWeaverLoginModule successfully initialized.");

   }

   @Override
   public boolean login(Map<String, Object> authenticationData) {
      USERNAME = (String) authenticationData.get("username");
```

```
        PASSWORD = (String) authenticationData.get("password");

        logger.info("SAPNetWeaverLoginModule login method called.");


        JCoCustomDestination customdestination = null;

        try {
          customdestination =
JCoDestinationManager.getDestination(DESTINATION).createCustomDestination();

          customdestination.getUserLogonData().setUser(USERNAME);
          customdestination.getUserLogonData().setPassword(PASSWORD);

          logger.info("Attributes:");
          logger.info(customdestination.getAttributes().toString());

            SAPTICKET = customdestination.getAttributes().getSSOTicket();


        } catch (JCoException e) {
            // TODO Auto-generated catch block
            throw new RuntimeException(e.getMessage());
        }

        return true;

    }


    @Override
    public UserIdentity createIdentity(String loginModule) {
        logger.info("SAPNetWeaverLoginModule createIdentity method called.");
        HashMap<String, Object> customAttributes = new HashMap<String, Object>();
        customAttributes.put("AuthenticationDate", new Date());
        customAttributes.put("SAPLoginTicket", SAPTICKET);

        UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null,
customAttributes, PASSWORD);
        return identity;
    }

    @Override
    public void logout() {
        logger.info("SAPNetWeaverLoginModule logout method called.");

        USERNAME = null;
        PASSWORD = null;
    }

    @Override
    public void abort() {
        logger.info("SAPNetWeaverLoginModule abort method called.");


        USERNAME = null;
        PASSWORD = null;
    }

    @Override
    public SAPNetWeaverLoginModule clone() throws CloneNotSupportedException {
        return (SAPNetWeaverLoginModule) super.clone();
    }


   private void readABAPConfiguration(Map<String, String> options) throws
MissingConfigurationOptionException {
      logger.info("Reading ABAP configuration values.");

      if (options.containsKey("hostname"))
         SAPHOSTNAME = options.get("hostname");
      else
         throw new MissingConfigurationOptionException("hostname");

      if (options.containsKey("sysnr"))
         SAPSYSNR = options.get("sysnr");
      else
         throw new MissingConfigurationOptionException("sysnr");

      if (options.containsKey("client"))
         SAPCLIENT = options.get("client");
      else
         throw new MissingConfigurationOptionException("client");

      if (options.containsKey("sid"))
         SAPSID = options.get("sid");
      else
         throw new MissingConfigurationOptionException("sid");

      DESTINATION = SAPSID + SAPCLIENT;

      Properties clientProperties = new Properties();

        clientProperties.setProperty(DestinationDataProvider.JCO_ASHOST, SAPHOSTNAME);
        clientProperties.setProperty(DestinationDataProvider.JCO_SYSNR,  SAPSYSNR);
        clientProperties.setProperty(DestinationDataProvider.JCO_CLIENT, SAPCLIENT);
        clientProperties.setProperty(DestinationDataProvider.JCO_LANG,   "en");
        clientProperties.setProperty(DestinationDataProvider.JCO_GETSSO2, "1");
        clientProperties.setProperty(DestinationDataProvider.JCO_GWHOST, SAPHOSTNAME);
        clientProperties.setProperty(DestinationDataProvider.JCO_GWSERV, "sapgw" +
SAPSYSNR);

        try
        {
        DynamicDestinationDataProvider myProvider = new
DynamicDestinationDataProvider(clientProperties, DESTINATION);
        com.sap.conn.jco.ext.Environment.registerDestinationDataProvider(myProvider);

        }catch (Exception e)
        {
        logger.info("LoginModule error:" + e.getMessage());
        System.err.println("LoginModule error:" + e.getMessage());
        }


        logger.info("LoginModule ABAP parameters successfully set.");
   }
   }
```

Tip: Verify at the operating system level, where the IBM MobileFirst Platform is running, that the `/etc/services` file contains entries for your SAP gateway instance (for example, `sapgw25 3325/tcp`).

The Java class that implements the login module must be created in the MobileFirst project, folder `server/java`.

Implement the SAP Java Connector destination data provider

The SAP Java Connector enables Java components to interact with SAP ABAP-based ERP systems. An implementation technique is to implement a custom destination data provider to set up the connection to the SAP system. Using a custom destination data provider enables the developer to set the credentials dynamically, which is the approach that is required for the login module in this scenario.

An important aspect is to configure the destination data provider to issue a login ticket when the login is successful.

The dynamic destination data provider implements the standard SAP JCO class `DestinationDataProvider` and overrides the mandatory methods, as shown in Example 1-17.

Example 1-17 Dynamic destination data provider implementation

```
public class DynamicDestinationDataProvider implements DestinationDataProvider {

   Map<String, Properties> propertiesForDestinationName = new HashMap<String,
Properties>();

   public DynamicDestinationDataProvider(Properties clientProperties, String
destinationName) {

         addDestination(destinationName, clientProperties);
      }

   public void addDestination( String destinationName, Properties properties )
    {
        propertiesForDestinationName.put( destinationName, properties );
    }
   @Override
   public Properties getDestinationProperties(String destinationName) {

   if ( propertiesForDestinationName.containsKey( destinationName ) )
    {
      return propertiesForDestinationName.get( destinationName );
    }
   else
   {
        throw new RuntimeException( "JCo destination not found: " +
destinationName );
    }
   }
   @Override
   public void setDestinationDataEventListener(
         DestinationDataEventListener arg0) {
      // TODO Auto-generated method stub
   }
   @Override
   public boolean supportsEvents() {
      // TODO Auto-generated method stub
      return false;
   }
}
```

The Java class that implements the destination data provider must be created in the MobileFirst project, folder `server/java`.

1.3 Testing the mobile application

This section describes how to test the mobile application and shows how the login module logic works within a mobile application.

Testing the mobile application involves the following three steps that this section describes:

1. Build the mobile application in MobileFirst Studio.

   Start the build process in IBM MobileFirst Studio and generate an executable mobile application.
2. Deploy the mobile application to the MobileFirst Development Server.
3. Run the mobile application and test the functionality in a standard web browser.

1.3.1 Build the mobile application in IBM MobileFirst Studio

The IBM MobileFirst Studio includes the capability to start the build procedure, which generates a deployable artifact. To build the mobile application in this scenario, complete the following steps:

1. Open MobileFirst Studio.
2. Expand the mobile project structure and right-click the mobile application folder.
3. In the context menu, select **Run As** → **2 Build All Environments** (Figure 1-7 on page 22).

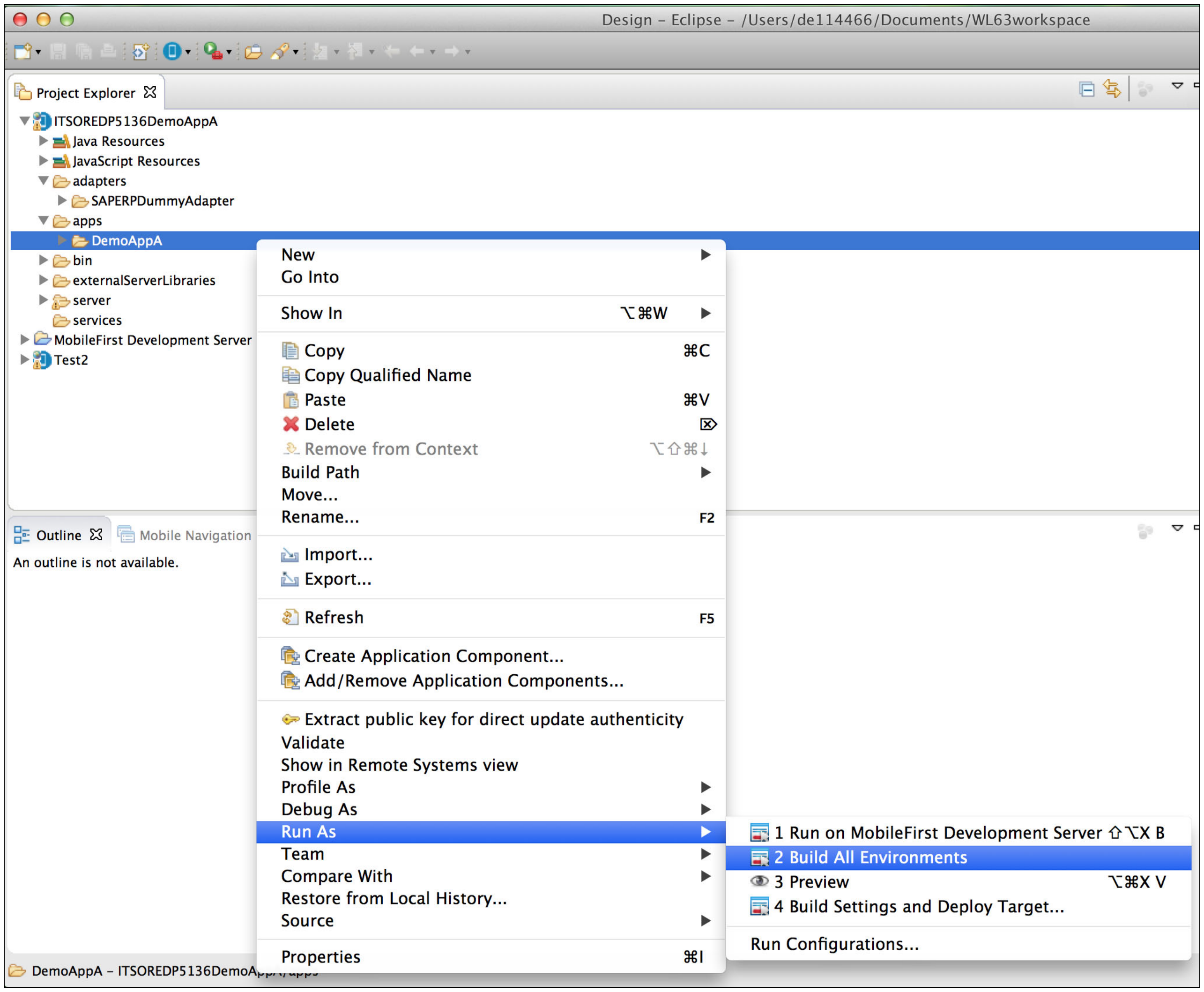


Figure 1-7 Start the build process through the context menu

4. Verify that no errors are logged in the Console view.

The build process is complete.

1.3.2 Deploy the mobile application to the IBM MobileFirst Development Server

MobileFirst Studio includes a fully functional MobileFirst Server that developers can use to locally develop, deploy, and test mobile applications. To deploy the mobile application in this scenario, complete the following steps:

1. Open **MobileFirst Studio**.
2. Expand the mobile project structure and right-click the mobile application folder.
3. In the context menu, select **Run As** → **1 Run on MobileFirst Development Server**.
4. Verify that no errors are logged in the Console view.

5. Expand the mobile project structure and right-click the **SAPERPDummyAdapter** folder (created in 1.2.2, “Create a new MobileFirst adapter in the hybrid application project” on page 6).
6. In the context menu, select **Run As** → **1 Deploy MobileFirst Adapter** (Figure 1-8).

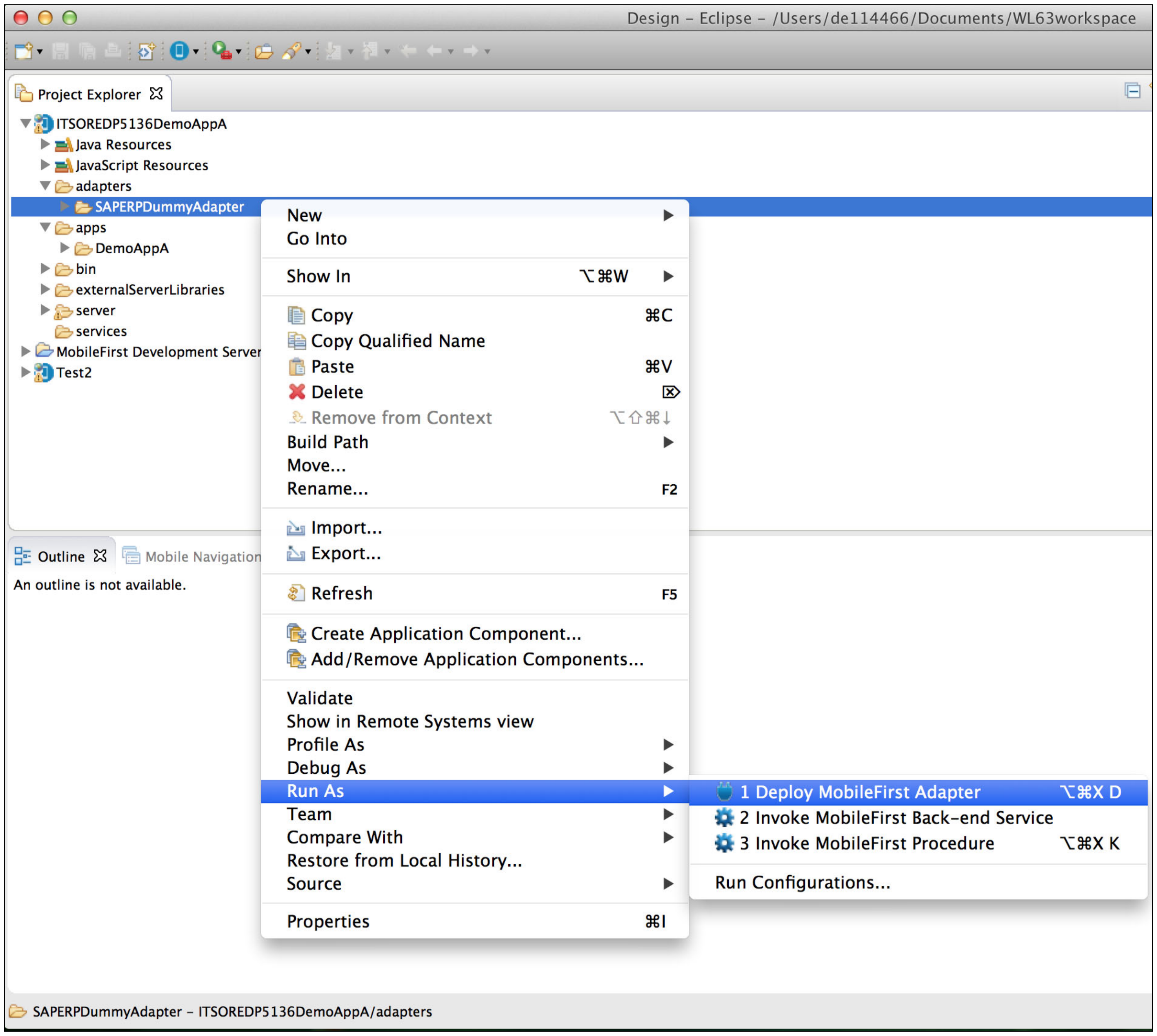
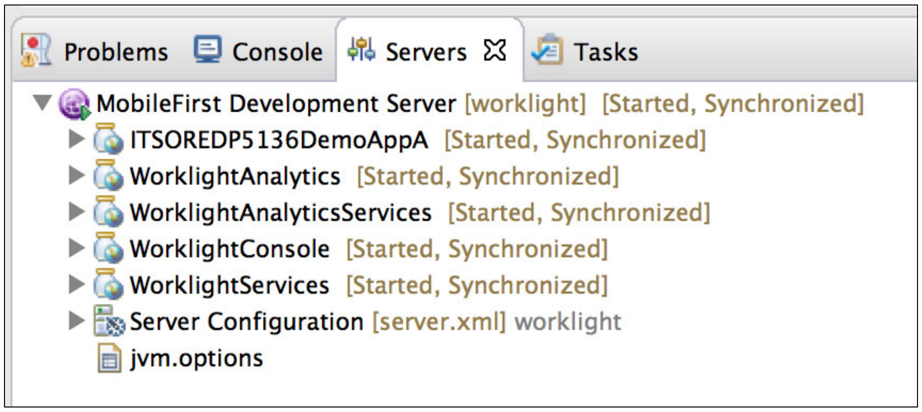


Figure 1-8 Deploy the MobileFirst adapter

7. Switch to the Servers view and verify that all deployments were successful (Figure 1-9).



MobileFirst Development Server [worklight] [Started, Synchronized]
ITSOREDP5136DemoAppA [Started, Synchronized]
WorklightAnalytics [Started, Synchronized]
WorklightAnalyticsServices [Started, Synchronized]
WorklightConsole [Started, Synchronized]
WorklightServices [Started, Synchronized]
Server Configuration [server.xml] worklight
jvm.options

Figure 1-9 Verify successful deployment of mobile application to MobileFirst Development Server

Now, the mobile application is deployed to the MobileFirst Development Server and ready for testing.

1.3.3 Run the mobile application

The mobile application can be tested with a standard web browser. In this scenario, the Chrome browser is used to open the MobileFirst Console and to start the mobile application.

This application has two test cases:

- A positive test case: Valid credentials are provided and the mobile application displays the secured business data.
- A negative test case: Invalid credentials are provided, and the mobile application displays an error message and prohibits the business data visualization.

Run a positive test case

In the positive test case, the user provides valid credentials in the form of a user ID and a password. The developed custom login module logic collects these credentials and passes them to the connected SAP Business Suite instance. The SAP system accepts the credentials and generates a valid Logon Ticket for this user. The developed custom login module logic puts the Logon Ticket into the security realm for future usage and displays the secrete business data to the user.

To run a positive test case, complete the following steps:

1. Open MobileFirst Studio.
2. Right-click the mobile project **ITSOREDP5136DemoAppA** and select **Open MobileFirst Console** (Figure 1-10 on page 25).

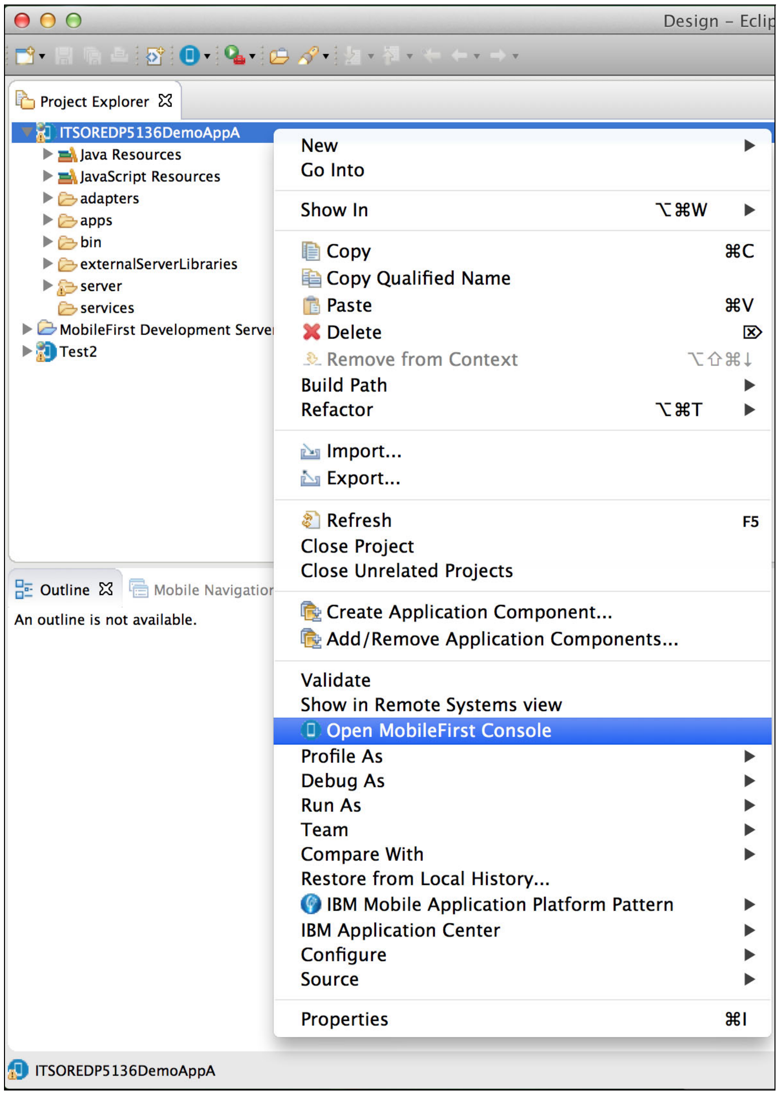


Figure 1-10 Open the MobileFirst Console

3. Click **Preview as Common Resources** (Figure 1-11).

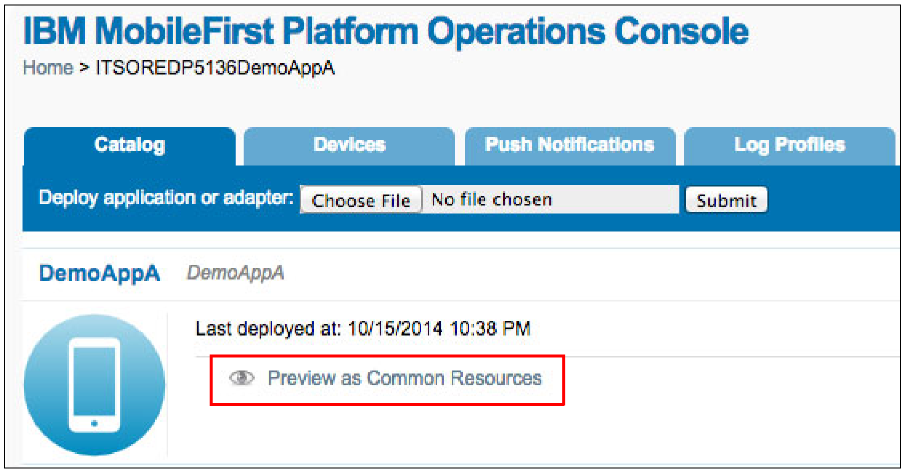
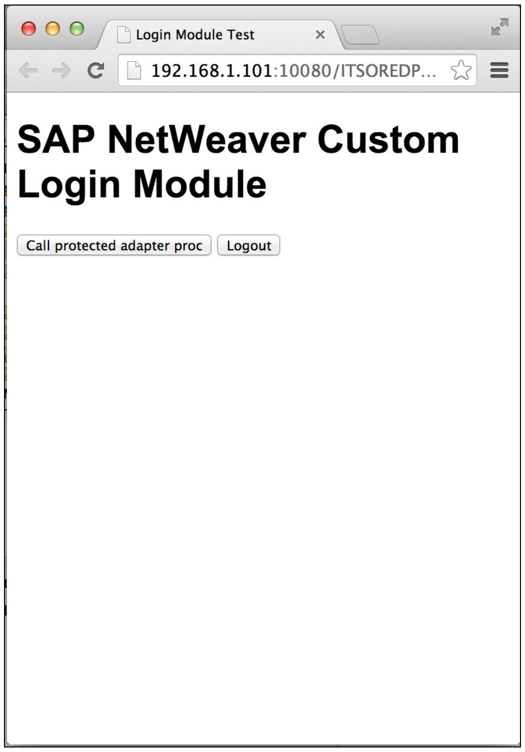

IBM MobileFirst Platform Operations Console
Home > ITSOREDP5136DemoAppA
Catalog
Devices
Push Notifications
Log Profiles
Deploy application or adapter:
Choose File
No file chosen
Submit
DemoAppA
DemoAppA
Last deployed at: 10/15/2014 10:38 PM
Preview as Common Resources

Figure 1-11 Preview mobile application as common resources

4. A browser window opens; it shows the start screen of the mobile app (Figure 1-12). Click **Call protected adapter proc**.

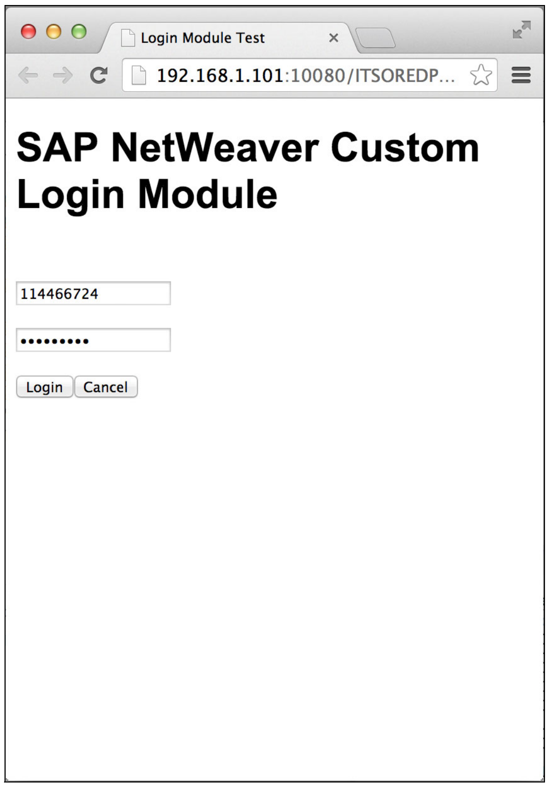


SAP NetWeaver Custom Login Module

Figure 1-12 Call protected adapter resource

5. The application displays the login form.

6. Enter a valid SAP user ID and password and click **Login** (Figure 1-13).



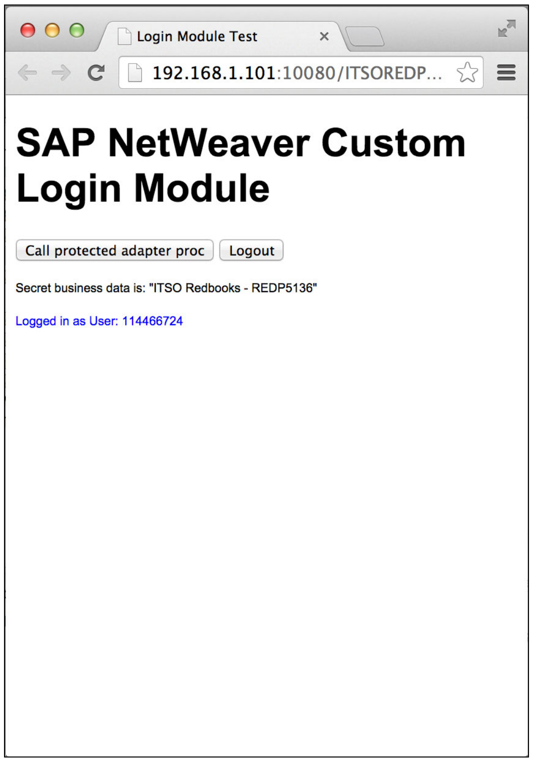
SAP NetWeaver Custom Login Module

114466724

•••••••••

Figure 1-13 Enter valid credentials

7. The system validates the credentials and the application displays the secret business data as shown in Figure 1-14.



SAP NetWeaver Custom Login Module

Secret business data is: "ITSO Redbooks - REDP5136"

Logged in as User: 114466724

Figure 1-14 Mobile application displays secured data after successful login

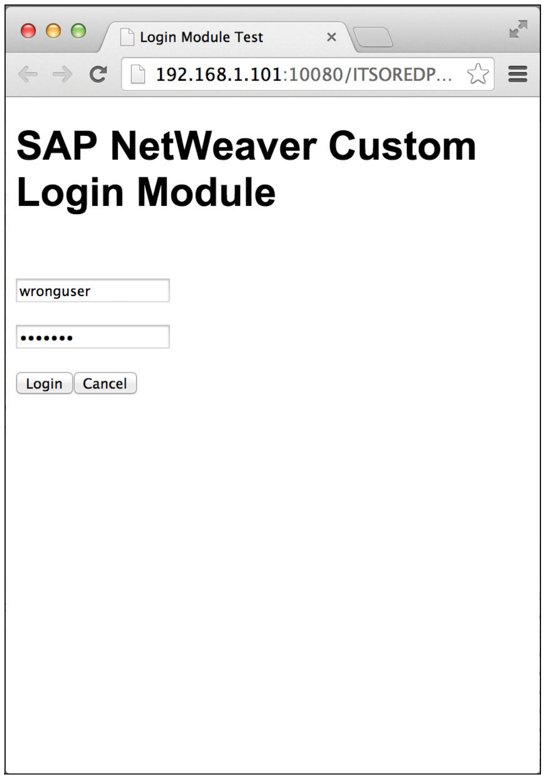
Now, the positive test case is complete. Clicking **Call protected adapter proc** again does not show the login form again because the MobileFirst Server recognizes that the user is already logged in. You can click **Logout** and run the test again.

Run a negative test case

In the negative test case, the user provides invalid credentials in the form of a wrong user ID or a wrong password. The developed custom login module logic collects these credentials and passes them to the connected SAP Business Suite instance. The SAP system rejects these credentials and provides a negative response to the login module which display an error message to the user without showing the requested secured business data.

To run a negative test, complete the following steps:

1. Repeat steps 1 on page 24 through 5 on page 26.
2. In the login form, enter invalid credentials and click **Login** (Figure 1-15).



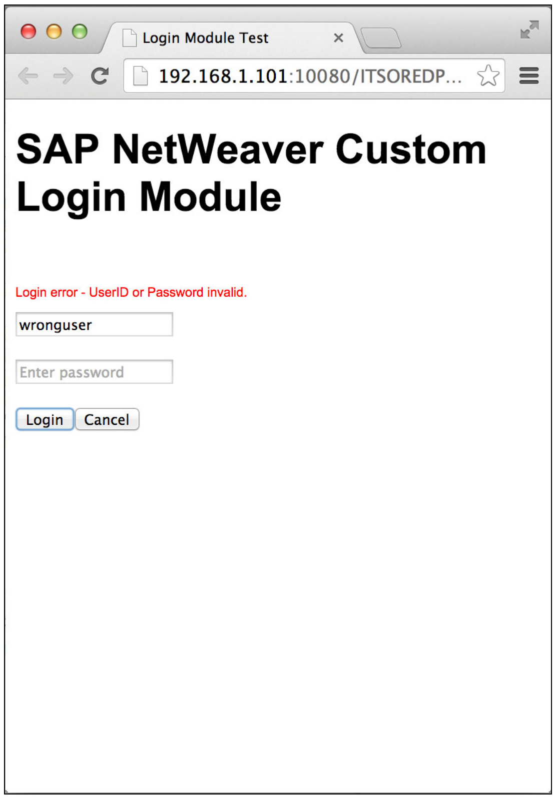
SAP NetWeaver Custom Login Module

wronguser

•••••••

Figure 1-15 Enter invalid credentials

3. The system recognizes that the credentials are not valid and displays an error message (Figure 1-16).



SAP NetWeaver Custom Login Module

Login error - UserID or Password invalid.

wronguser

Enter password

Figure 1-16 Mobile application displays error message

The negative test case is complete.

2

Mobile application single sign-on for SAP business systems

This chapter describes how to implement single sign-on (SSO) by enhancing the scenario described in Chapter 1, “Mobile authentication with SAP Business Suite” on page 1.

This chapter describes two approaches to implement SSO when integrating with SAP Business Suite:

- Using the IBM MobileFirst Cast Iron adapter
- Using the IBM MobileFirst SAP NetWeaver Gateway adapter

2.1 IBM MobileFirst Platform Foundation SSO capabilities

IBM MobileFirst Platform Foundation provides secure, end-to-end communication by positioning a server that oversees the flow of data between the mobile application and your back-end systems. With IBM MobileFirst Platform Foundation, you can define custom security handlers for any access to this flow of data. Because any access to data of a mobile application must go through this server instance, you can define different security handlers for mobile applications, web applications, and back-end access. With this kind of granular security, you can define separate levels of authentication for different functions of your mobile application, or avoid sensitive information to be accessed from a mobile application entirely.

IBM MobileFirst Platform Foundation provides a rich and enhanced security framework to mobile application developers. The security framework contains a large set of pre-built security capabilities that can be leveraged by developers with only configuration, rather than developing these features manually. These security features will be extended, enhanced, and hardened over time, giving the developer the ability to focus on the core mobile application development.

One key aspect of the IBM MobileFirst Platform security framework is to provide single sign-on (SSO) capabilities to mobile application developer. SSO enables users to access multiple resources within an application by authenticating only once. This capability is especially important for mobile applications that display business data that originates from various back-end systems. The built-in SSO integration capabilities in the MobileFirst security framework are a key feature of this mobile enterprise application platform.

When a user successfully logs in through an SSO-enabled login module, the user gains access to all the resources that are using the same login module, without having to authenticate again to each of them. The authenticated state remains valid as long as requests to resources protected by the login module are being issued within the timeout period, which is identical to the session timeout period.

SSO is a generic concept; the technical details to achieve SSO behavior must be identified and verified for each scenario. In general, implementing SSO in any scenario is feasible, but how much custom coding is needed to do so must be evaluated carefully.

For complex security scenarios, the advisable approach is to evaluate the usage of a central security component to manage an efficient security topology. IBM MobileFirst Platform Foundation is able to incorporate additional security components such as IBM DataPower® Gateway and IBM Security Access Manager or any other non IBM popular security devices or components.

An SSO solution is always specific to the particular scenario and it is heavily influenced by the parties involved and the general requirements of the SSO integration defined for the scenario. This chapter describes two approaches for an SSO solution:

- The first example uses the IBM MobileFirst Cast Iron adapter to call an IBM Cast Iron orchestration running in the cloud. The adapter uses basic authentication to trigger the call to IBM Cast Iron orchestration and passes a valid SAP Logon Ticket in a custom HTTP header. This Logon Ticket is used by the orchestration to interact with the SAP back-end system as the SAP user that is represented by the Logon Ticket. This approach is also known as *named user integration pattern*.
- The second example uses the IBM MobileFirst SAP NetWeaver Gateway adapter to call an SAP NetWeaver Gateway service. In this example, the MobileFirst security framework passes the credentials of the logged-in user automatically to the called SAP NetWeaver Gateway server. There is no special coding needed in the mobile application.

The decision of which adapter and which SSO approach to use depends on the existing infrastructure and business scenario. If the organization has an SAP NetWeaver Gateway in place, leveraging it is a good idea. If no SAP NetWeaver Gateway is in place or if the SAP NetWeaver Gateway does not provide the required business functions, consider using the Cast Iron adapter to accomplish the integration.

The decision of which SSO approach to use, technical user integration pattern or named user integration pattern, depends on the business scenario. The technical user approach is easier to maintain from an operational perspective but the organization might have the requirement to run all interactions with the SAP system under the security context of the business user, in which case the named user pattern must be used.

2.2 Using the IBM MobileFirst Cast Iron adapter for SSO authentication with SAP Business Suite

The Cast Iron SAP connector supports the most popular interaction patterns with the SAP system from a security perspective by using a technical user identity, a named user identity, or SAP Logon Tickets. IBM MobileFirst includes a pre-built adapter for Cast Iron. This adapter enables the mobile application developer to easily incorporate any available Cast Iron orchestration. For a description of the overall architecture, benefits, and challenges when using this integration approach, see the “Mobile access for SAP” topic in *IBM Software for SAP Solutions*, SG24-8230.

This section explains how to enhance the scenario that is described in Chapter 1, “Mobile authentication with SAP Business Suite” on page 1. The enhancement is to use a MobileFirst Cast Iron adapter instead of the MobileFirst adapter, `SAPERPDummyAdapter`, that is created in 1.2.2, “Create a new MobileFirst adapter in the hybrid application project” on page 6. The objective of this scenario is to implement the connectivity with the SAP system using SSO between the MobileFirst adapter and the SAP back-end system.

The MobileFirst Cast Iron adapter calls an IBM Cast Iron orchestration and passes a valid SAP Logon Ticket to this orchestration. The IBM Cast Iron orchestration uses the SAP Logon Ticket to authenticate with the SAP Business Suite and calls an SAP business function. For demonstration purposes in this scenario, the Cast Iron orchestration calls `STFC_CONNECTION`, which is a simple SAP function module. This function module takes only one string parameter as input. It returns this string as a response plus a second response parameter that contains SAP system-specific details with the format that is shown in Example 2-1.

Example 2-1 Sample output of SAP field RESPTEXT

```
Pattern:
<SAP release>  <SAP SystemID> <SAP Date>  <SAP Time>  <SAP Client/UserID/Language>

Example:
SAP R/3 Rel. 702   Sysid: BMU Date: 20141103   Time: 162318   Logon_Data:
200/113366724/E
```

This function module is well-suited to demonstrate the SSO capability because it gets only the SAP Logon Ticket and it returns the SAP user ID under which the function module was run.

2.2.1 IBM Cast Iron orchestration overview

The creation of the IBM Cast Iron orchestration is not the focus of this scenario; details about the creation of the orchestration used in this scenario are not included in this paper.

However, to understand the example in this scenario, understanding the basic functionality provided by the Cast Iron orchestration is important.

IBM Cast Iron is an efficient integration technology well-suited for mobile and cloud-based scenarios. IBM Cast Iron can integrate with a wide variety of back-end systems including applications in the SAP Business Suite.

A key advantage of the IBM Cast Iron technology is the graphical editor, which helps integration developers to more easily design powerful orchestrations (Figure 2-1)

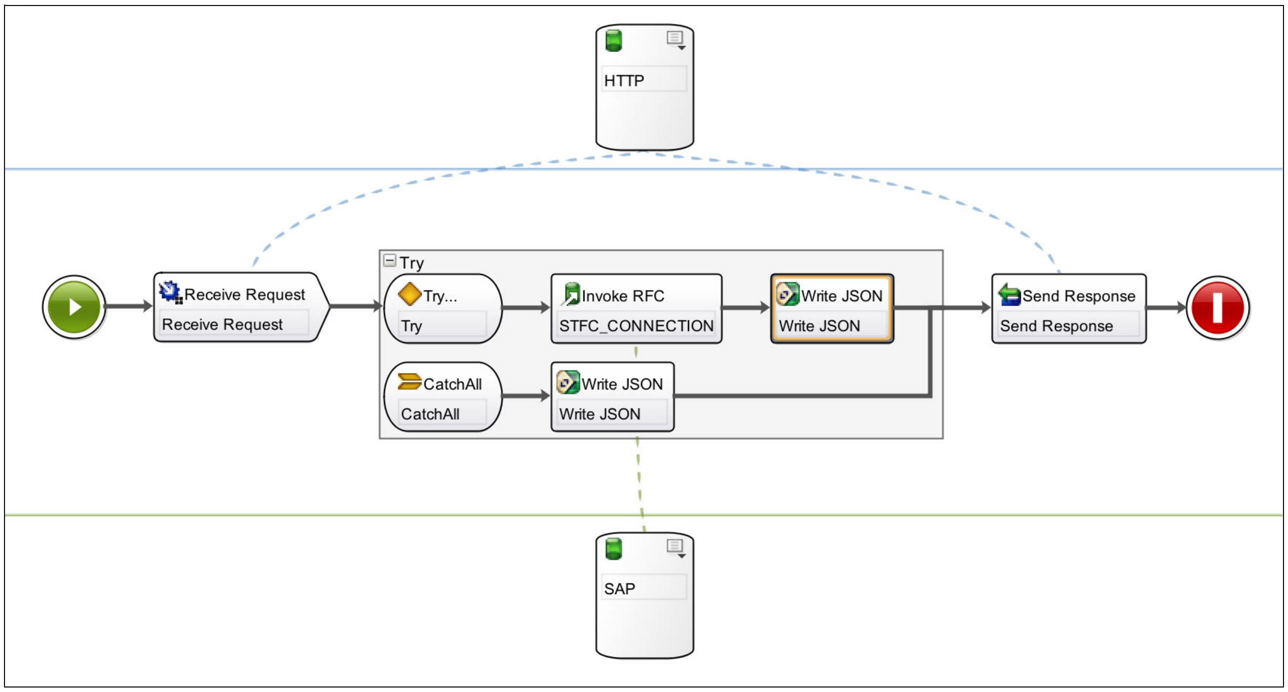
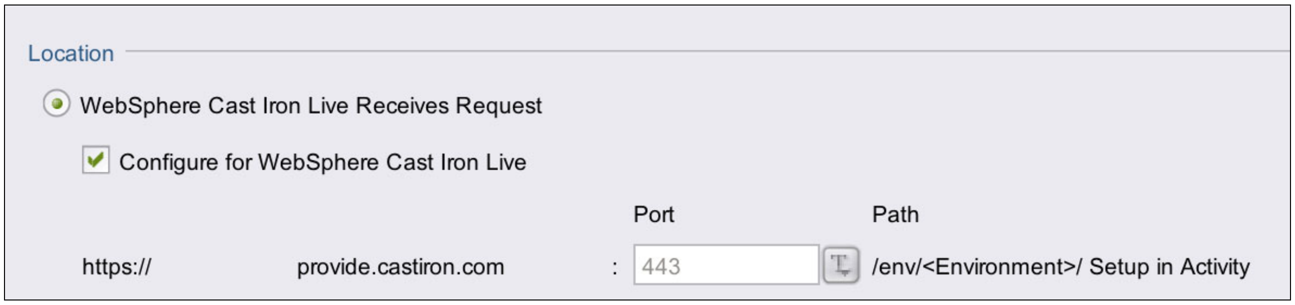


Figure 2-1 IBM Cast Iron orchestration calling SAP back-end system

The HTTP protocol configuration object defines which ports are valid for this orchestration.

In this example, the orchestration runs in the cloud (IBM WebSphere Cast Iron Live) and the settings are configured as shown in Figure 2-2.



Location

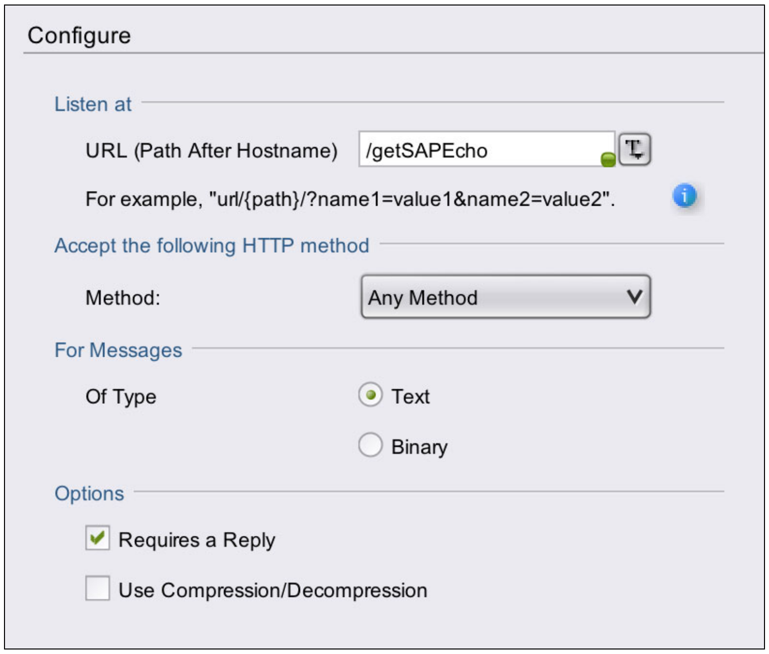
● WebSphere Cast Iron Live Receives Request

✓ Configure for WebSphere Cast Iron Live

		Port	Path
https://	provide.castiron.com :	443	/env/<Environment>/ Setup in Activity

Figure 2-2 Orchestration is configured to run in the cloud

The Cast Iron platform routes incoming requests to the proper orchestration based on the context URI definition of the `Receive Request` node as shown in Figure 2-3.



Configure

Listen at

URL (Path After Hostname) /getSAPEcho

For example, "url/{path}/?name1=value1&name2=value2".

Accept the following HTTP method

Method: Any Method

For Messages

Of Type [✓ Text] Binary

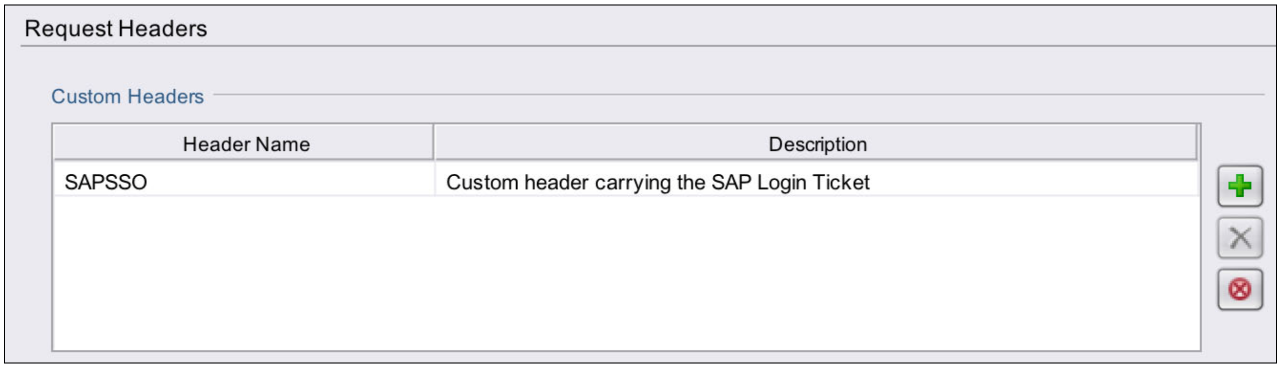
Options

[✓] Requires a Reply

[ ] Use Compression/Decompression

Figure 2-3 Define context URI for HTTP routing

The received HTTP request is handled by the orchestration. In this scenario, a custom HTTP header field is used to carry the SAP Logon Ticket (Figure 2-4).



Request Headers

Custom Headers

Header Name	Description
SAPSSO	Custom header carrying the SAP Login Ticket

Figure 2-4 Custom HTTP header field

The IBM Cast Iron graphical editor enables the developer to easily map the Logon Ticket to the credentials area of the SAP connection properties structure (Figure 2-5).

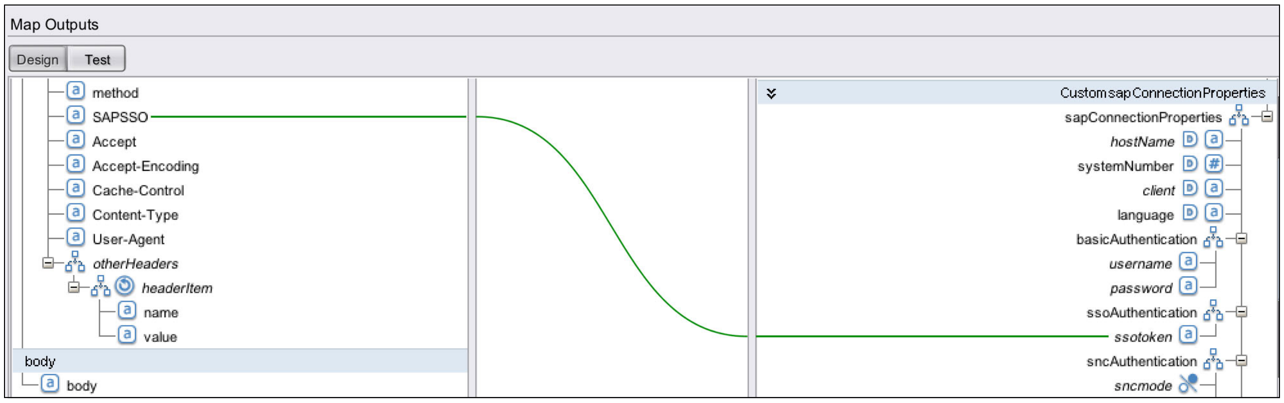


Figure 2-5   Prepare SAP interaction to use SAP Logon Ticket

Besides the connectivity type, usually business parameters are also mapped as shown in Figure 2-6.

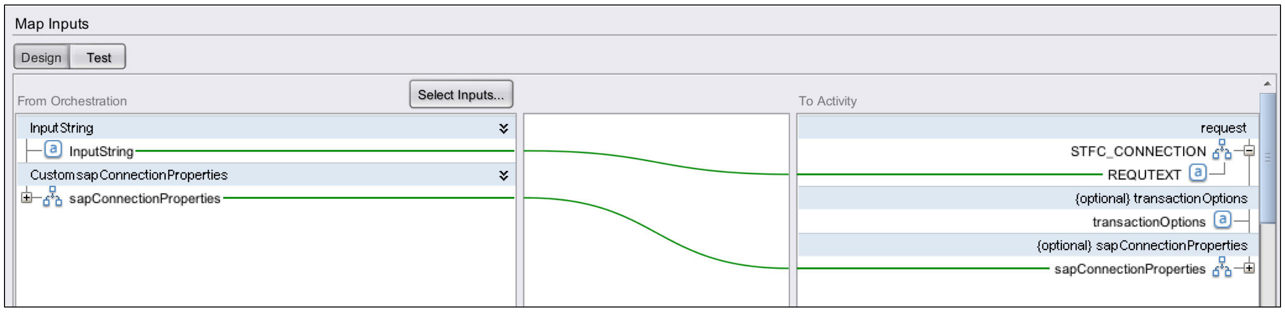


Figure 2-6   Data input mapping

The same mapping must be applied after the SAP call to prepare the `ResultSet` accordingly. The orchestration produces a JSON `ResultSet` as shown in Example 2-2.

Example 2-2   Positive response

```
{
   "null": "sp_0:STFC_CONNECTION.Response",
   "ECHOTEXT": "Hello Mr. Birkler",
   "RESPTEXT": "SAP R/3 Rel. 702   Sysid: BMU Date: 20141103   Time: 174343
Logon_Data: 200/113366724/D"
}
```

If the provided SAP Logon Ticket is not accepted by the SAP system that is being called (negative path), the Cast Iron orchestration in this scenario returns a negative response as shown in Example 2-3.

Example 2-3 Negative response to invalid SAP Logon Ticket

```
{"null": {
   "ns2:name": "sap.Invoke RFC.operation",
   "message": "An error occurred while trying to execute activity RFC. Error is: 
javax.resource.ResourceException: 
com.ibm.j2ca.base.exceptions.AuthenticationFailException: System received an 
expired SSO ticket on <hostname> sysnr <sapsysnr>",
   "activityId": 6,
   "activityName": "STFC_CONNECTION",
   "faultTime": "2014-11-03T16:05:55.026Z"
}}
```

The negative path can have a second response when no SAP Logon Ticket is provided, which results in the response shown in Example 2-4.

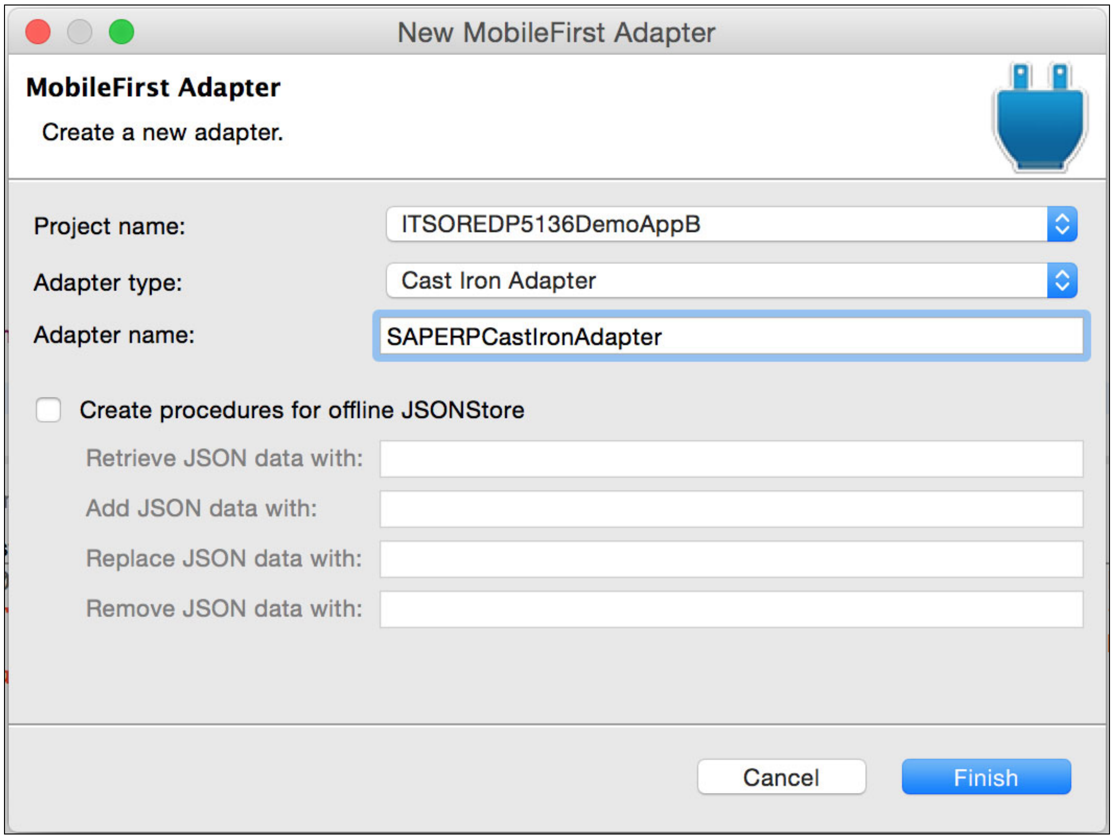
Example 2-4 Negative response when SAP Logon Ticket is missing

```
{"null":{
   "ns2:name":"sap.Invoke RFC.operation",
   "message":"An error occurred while trying to execute activity RFC. Error is: 
javax.resource.ResourceException: 
javax.resource.ResourceException: 
Exception in connecting to SAP : Configuration of destination <hostname>.946220 is 
incomplete: 
Parameter containing a user ID is missing: neither user nor user alias nor 
external ID nor SSO ticket nor X.509 certificate is 
specified","activityId":6,"activityName":"STFC_CONNECTION","faultTime":"2014-11-03
T15:37:34.018Z"}}
```

2.2.2 Develop an IBM MobileFirst Cast Iron adapter

Create a new MobileFirst adapter by using the corresponding Eclipse wizard in MobileFirst Studio as follows:

1. Select **MobileFirst Adapter**.
2. Select **Cast Iron Adapter** as the adapter type, provide an adapter name (`SAPERPCastIronAdapter` in this example), and click **Finish** (Figure 2-7).


New MobileFirst Adapter

MobileFirst Adapter

Create a new adapter.

Project name: ITSOREDP5136DemoAppB

Adapter type: Cast Iron Adapter

Adapter name: SAPERPCastIronAdapter

☐ Create procedures for offline JSONStore

Retrieve JSON data with:

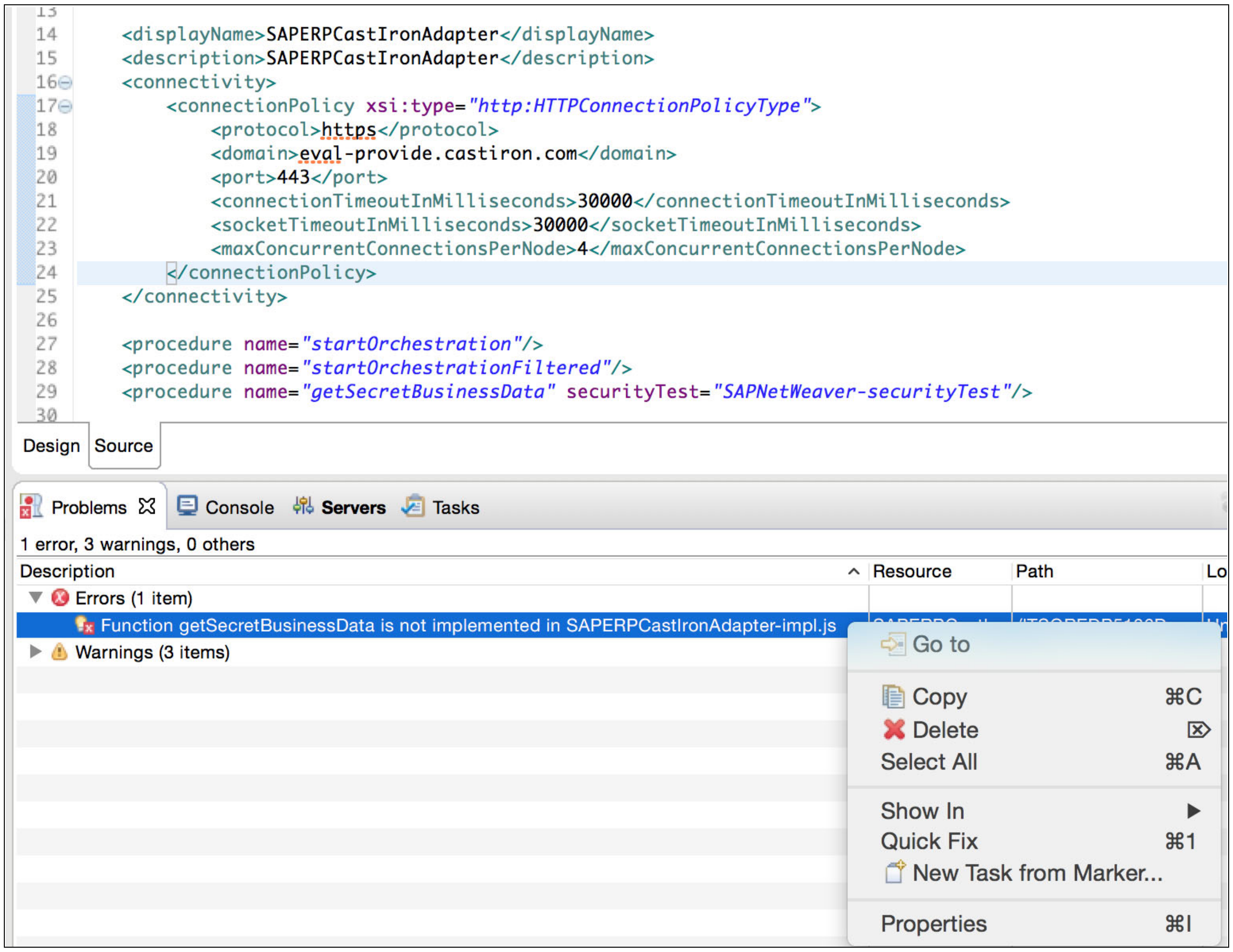
Add JSON data with:

Replace JSON data with:

Remove JSON data with:

Figure 2-7 Creating the IBM MobileFirst Cast Iron adapter

3. Open the `SAPERPCastIronAdapter.xml` file and modify it as shown in Figure 2-8.



```
    <displayName>SAPERPCastIronAdapter</displayName>
    <description>SAPERPCastIronAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>https</protocol>
            <domain>eval-provide.castiron.com</domain>
            <port>443</port>
            <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
            <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
            <maxConcurrentConnectionsPerNode>4</maxConcurrentConnectionsPerNode>
        </connectionPolicy>
    </connectivity>

    <procedure name="startOrchestration"/>
    <procedure name="startOrchestrationFiltered"/>
    <procedure name="getSecretBusinessData" securityTest="SAPNetWeaver-securityTest"/>
```

1 error, 3 warnings, 0 others

Description	Resource	Path	Lo
Errors (1 item)			
Function getSecretBusinessData is not implemented in SAPERPCastIronAdapter-impl.js			
Warnings (3 items)			

Figure 2-8 Modifying the IBM MobileFirst Cast Iron adapter XML file

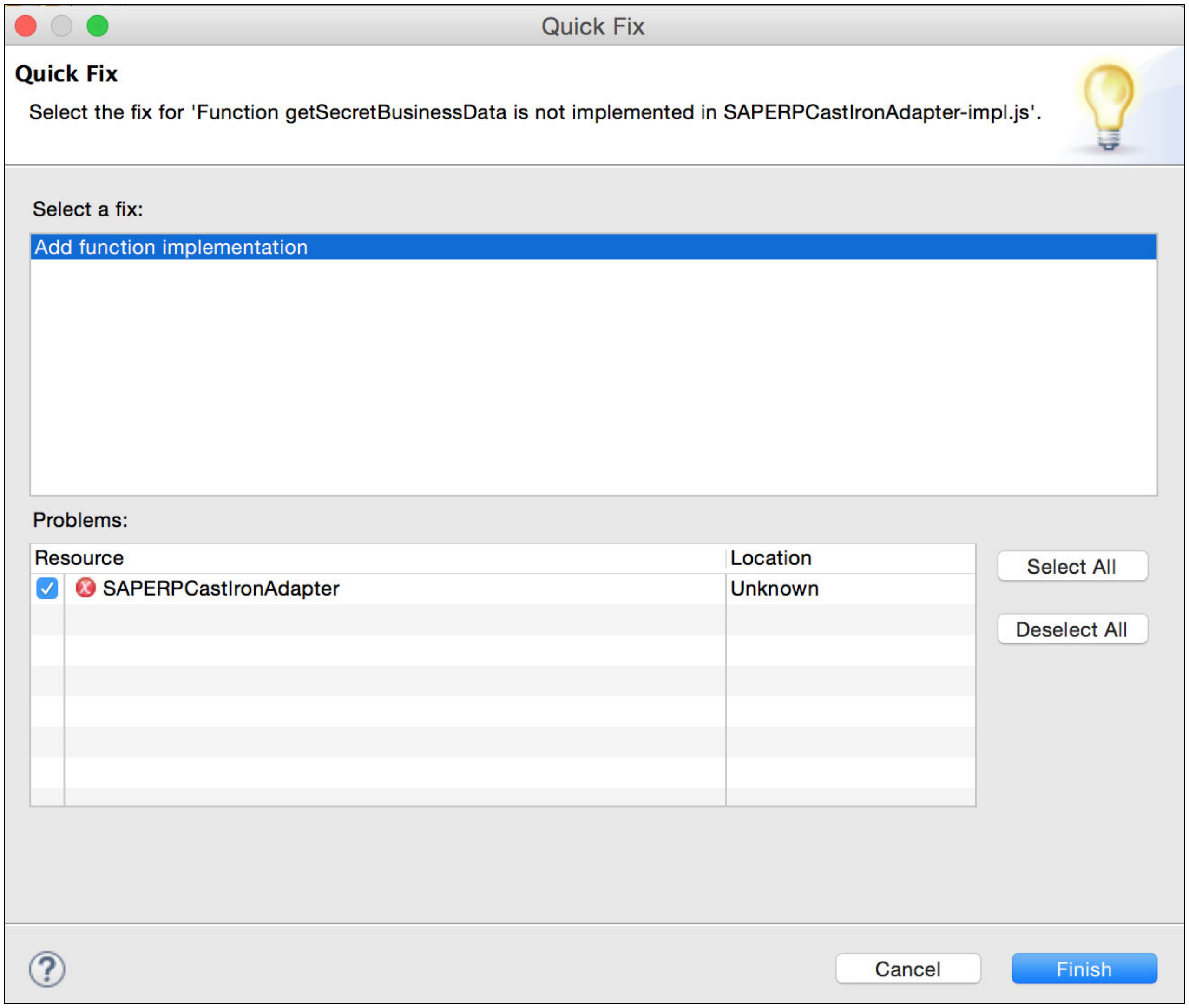
Notice the following information:

– The `connectionPolicy` attributes must point to the Cast Iron orchestration.

– The XML file must include a `procedure` tag named `getSecretBusinessData`, which is secured by the security test `SAPNetWeaver-securityTest`.

4. Save the `SAPERPCastIronAdapter.xml` file. The system shows an error because the JavaScript implementation file is missing the corresponding function.

5. Use Eclipse Quick Fix to add the missing JavaScript function to the adapter implementation file (Figure 2-9 on page 40).


Quick Fix

Quick Fix

Select the fix for 'Function getSecretBusinessData is not implemented in SAPERPCastIronAdapter-impl.js'.

Select a fix:

Add function implementation

Problems:

Resource	Location
✓ SAPERPCastIronAdapter	Unknown

Figure 2-9 Generating the JavaScript logic in adapter implementation file

6. Add the JavaScript code to the function body as shown in Example 2-5.

Example 2-5 Declaration of procedure getSecretBusinessData in SAPERPCastIronAdapter-impl.js

```
function getSecretBusinessData() {
   var sapticket = WL.Server.getActiveUser().attributes.SAPLoginTicket;
   var input = {
         method  : 'get',
         appName : 'myApp',
         headers: {Authorization: "Basic YWRW1peDEwAYmx1taW5ZLmlibwNGRlbTpQVOQW8=",
SAPSSO: sapticket},
         requestType: 'http',
         path : '/env/Development/getSAPEcho',
         returnedContentType : 'json'
   };
   var response = WL.Server.invokeCastIron(input);
   return { secretBusinessData: response };
}
```

7. Save the JavaScript file.

The first line of the JavaScript code accesses the security context of the server and uses the SAP Logon Ticket that was placed in the security realm by the custom login module, which is developed in Chapter 1, “Mobile authentication with SAP Business Suite” on page 1.

The JavaScript code uses two HTTP header fields to transport the data that is relevant to the authentication:

- The `Authorization` header carries a basic authentication header which is used to authenticate the MobileFirst adapter with the Cast Iron orchestration.
- The `SAPSSO` header carries the SAP Logon Ticket which is used by the Cast Iron orchestration to make the SAP call with the identity encoded in the ticket.

2.2.3 Implement the IBM Cast Iron adapter in the hybrid sample application

The `main.js` JavaScript file includes application-specific code. In this scenario, the file contains the JavaScript function to call the MobileFirst Cast Iron adapter.

Implement the MobileFirst Cast Iron adapter in the mobile application as follows:

1. Open the `main.js` file.
2. Add the code snippet, shown in Example 2-6, to invoke the adapter procedure to receive the back-end data.

Example 2-6 Calling the adapter procedure to receive back-end data

```
function getSecretBusinessData(){
   var invocationData = {
         adapter: "SAPERPCastIronAdapter",
         procedure: "getSecretBusinessData",
         parameters: []
   };

   WL.Client.invokeProcedure(invocationData, {
      onSuccess: getSecretBusinessData_Callback,
      onFailure: getSecretBusinessData_Callback
   });
}
```

3. Add the code snippet, shown in Example 2-7, to implement a callback method to process the results.

Example 2-7 Implementing the callback function to process results

```
function getSecretBusinessData_Callback(response){
   if(response.invocationResult.isSuccessful) {

      document.getElementById('AppHomeResults').innerHTML = "Secret business
data is: <br>" +
JSON.stringify(response.invocationResult.secretBusinessData.RESPTEXT);

      if(response && (response.invocationResult) &&
(response.invocationResult['WL-Authentication-Success']) &&
(response.invocationResult['WL-Authentication-Success']['SAPNetWeaverRealm']))
      {
         var realm =
response.invocationResult['WL-Authentication-Success']['SAPNetWeaverRealm'];
```

```
            document.getElementById('AppHomeLoginDetails').innerHTML = "Logged in
as User: " + realm.userId;

        }
    }else
    {
            document.getElementById('AppHomeResults').innerHTML =
JSON.stringify(response);
        }
}
```

4. Save the `main.js` file.

2.2.4 Test the mobile application

The mobile application can be tested with a web browser. The Chrome browser was used in this scenario to open the IBM MobileFirst Console and to start the mobile application. The mobile application triggers the MobileFirst Cast Iron adapter procedure, which calls the corresponding IBM Cast Iron orchestration. The orchestration acts as an SAP client calling the SAP function module `STFC_CONNECTION`, which echoes the input parameter and returns system-specific information such as the logged-in user ID.

To test the mobile application, complete following steps:

1. Build the mobile application and deploy it to the IBM MobileFirst Studio Development Server by using the context menu as shown in Figure 2-10.

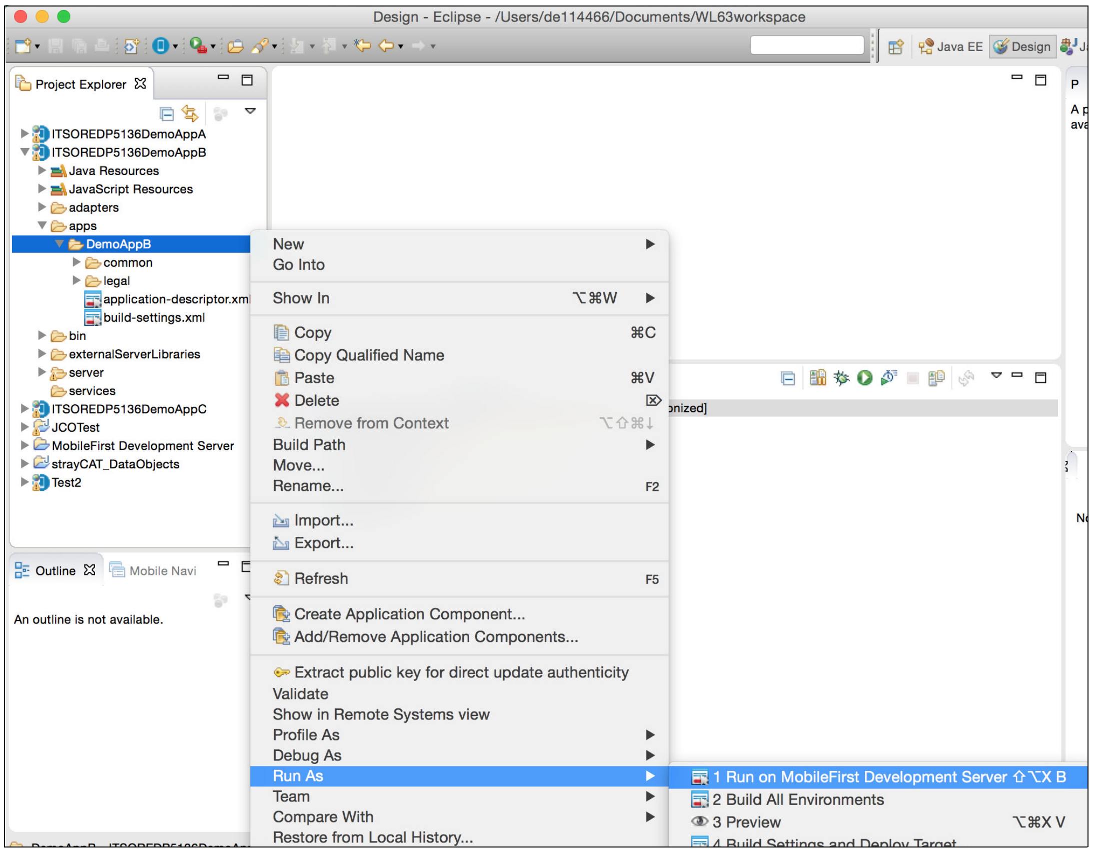


Figure 2-10 Build and run the mobile app in MobileFirst Development Studio Development Server

2. Select **Open MobileFirst Console** to open the console (Figure 2-11).

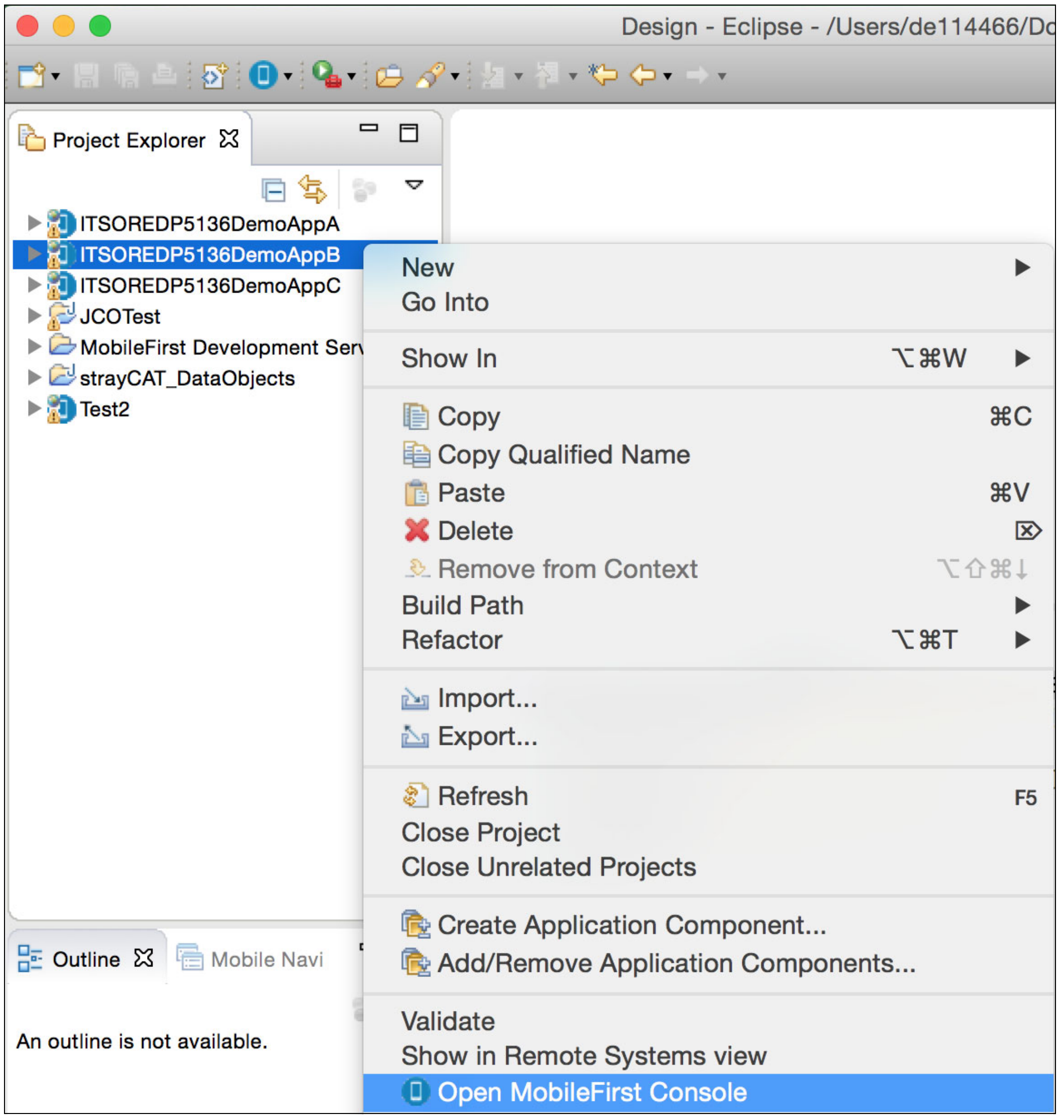
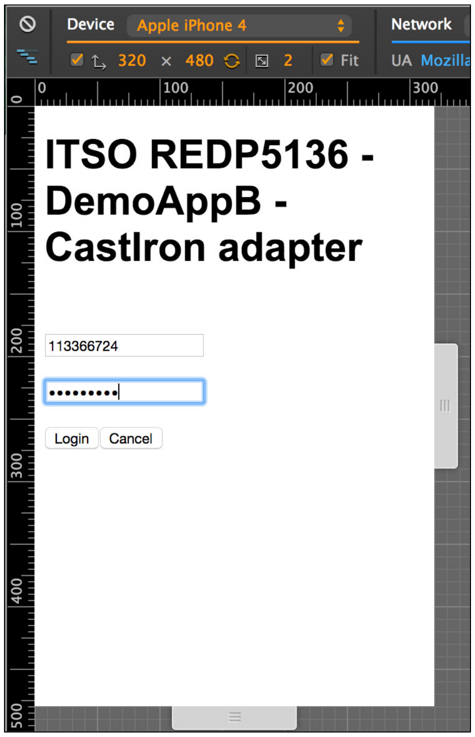


Figure 2-11 Open MobileFirst Console

3. Select **Preview as Common Resources**.
4. Click **Call protected procedure proc**.

5. Enter a valid SAP user ID and password into the login form and click **Login** (Figure 2-12).



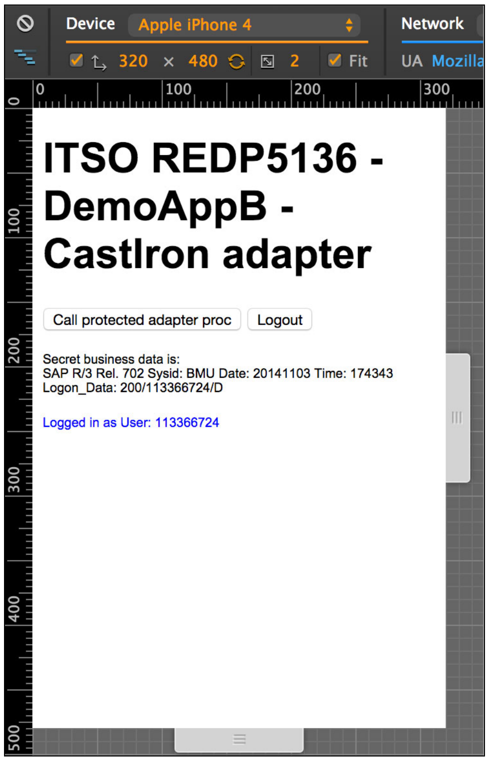
ITSO REDP5136 - DemoAppB - CastIron adapter

113366724

•••••••••

Figure 2-12 Enter valid credentials

The mobile application displays the results received from the SAP system (Figure 2-13). The `Logon_Data` section documents that the SAP Business Suite accepted the SAP Logon Ticket and identified the calling IBM Cast Iron orchestration as the SAP user 113366724.



ITSO REDP5136 - DemoAppB - CastIron adapter

Secret business data is:
SAP R/3 Rel. 702 Sysid: BMU Date: 20141103 Time: 174343
Logon_Data: 200/113366724/D

Logged in as User: 113366724

Figure 2-13 Secret business data received from SAP

2.3 Using the IBM MobileFirst SAP NetWeaver Gateway adapter for SSO authentication with SAP Business Suite

MobileFirst provides a ready-to-use adapter for SAP NetWeaver Gateway. This capability includes a wizard-driven code generation feature to perform dynamic introspection on an SAP NetWeaver Gateway instance, and to create the required artifacts automatically. The generated objects are packaged and deployed automatically within the MobileFirst environment. For a description of the overall architecture, benefits, and challenges when using this integration approach, see the “Mobile access for SAP” topic in *IBM Software for SAP Solutions*, SG24-8230.

This section explains how to enhance the scenario that is described in Chapter 1, “Mobile authentication with SAP Business Suite” on page 1. The enhancement is to use a MobileFirst SAP NetWeaver Gateway adapter instead of the adapter, `SAPERPDummyAdapter`, that is created in 1.2.2, “Create a new MobileFirst adapter in the hybrid application project” on page 6.

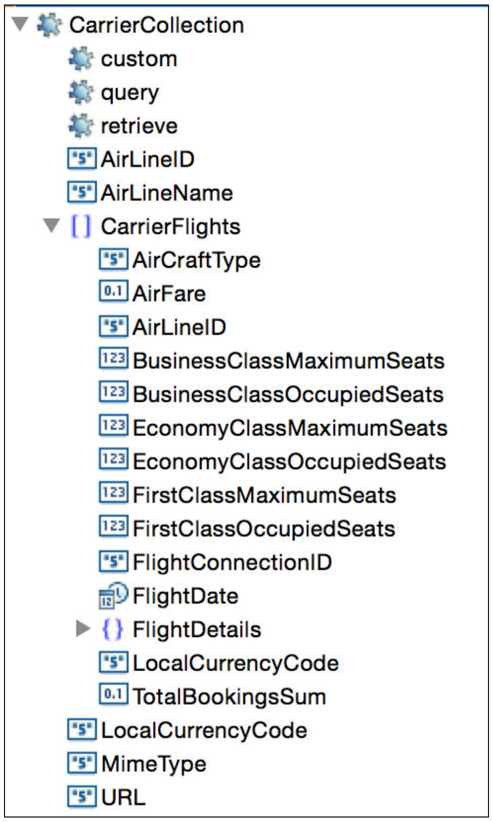
The MobileFirst SAP NetWeaver Gateway adapter calls SAP NetWeaver Gateway services and passes the validated credentials of the logged in security realm to the underlying SAP

NetWeaver Gateway server. The MobileFirst security framework controls transferring the credential through the adapter configuration file. There is no special coding required.

The example in this section demonstrates how to interact with an SAP NetWeaver Gateway service using the SSO capabilities of the IBM MobileFirst Platform security framework.

2.3.1 SAP NetWeaver Gateway service overview

This scenario uses a simple SAP NetWeaver Gateway service that is provided by SAP with the SAP Flight Data Application example. This scenario uses the service `RMTSAMPLEFLIGHT_2`. This service takes an airline code as input parameter and returns the full airline name as output parameter. The SAP NetWeaver Gateway service defines several collections; in this scenario, only the `CarrierCollection` is of interest and it has the structure shown in Figure 2-14.



```
CarrierCollection
    custom
    query
    retrieve
    AirLineID
    AirLineName
    CarrierFlights
        AirCraftType
        AirFare
        AirLineID
        BusinessClassMaximumSeats
        BusinessClassOccupiedSeats
        EconomyClassMaximumSeats
        EconomyClassOccupiedSeats
        FirstClassMaximumSeats
        FirstClassOccupiedSeats
        FlightConnectionID
        FlightDate
        FlightDetails
        LocalCurrencyCode
        TotalBookingsSum
    LocalCurrencyCode
    MimeType
    URL
```

Figure 2-14 SAP NetWeaver Gateway service interface structure

The MobileFirst adapter populates only a minimum data set and reads only the `AirLineName` parameter from the response object. The input and output parameters have the structure shown in Example 2-8.

Example 2-8 SAP NetWeaver Gateway service: Input and output parameters

```
Input:
{"keys" : {"AirLineID" : "LH"}}

Output:
{{"AirLineID" : "LH"}, {"AirLineName" : "Lufthansa"}}
```

The SA P NetWeaver Gateway service requires the calling client to authenticate by using basic authentication. Unauthenticated requests are nor served by the SAP NetWeaver Gateways service.

2.3.2 Develop a MobileFirst SAP NetWeaver Gateway adapter

The main difference between the SAP NetWeaver adapter and other MobileFirst adapters is that the SAP NetWeaver Gateway adapter is generated by using a discovery wizard. This component interacts with the SAP NetWeaver Gateway at design time and generates all needed artifacts automatically in the MobileFirst project structure.

To generate the adapter, complete the following steps:

1. Create a MobileFirst SAP NetWeaver Gateway adapter by using the **Discover Back-end Services** Eclipse wizard in MobileFirst Studio (Figure 2-15).

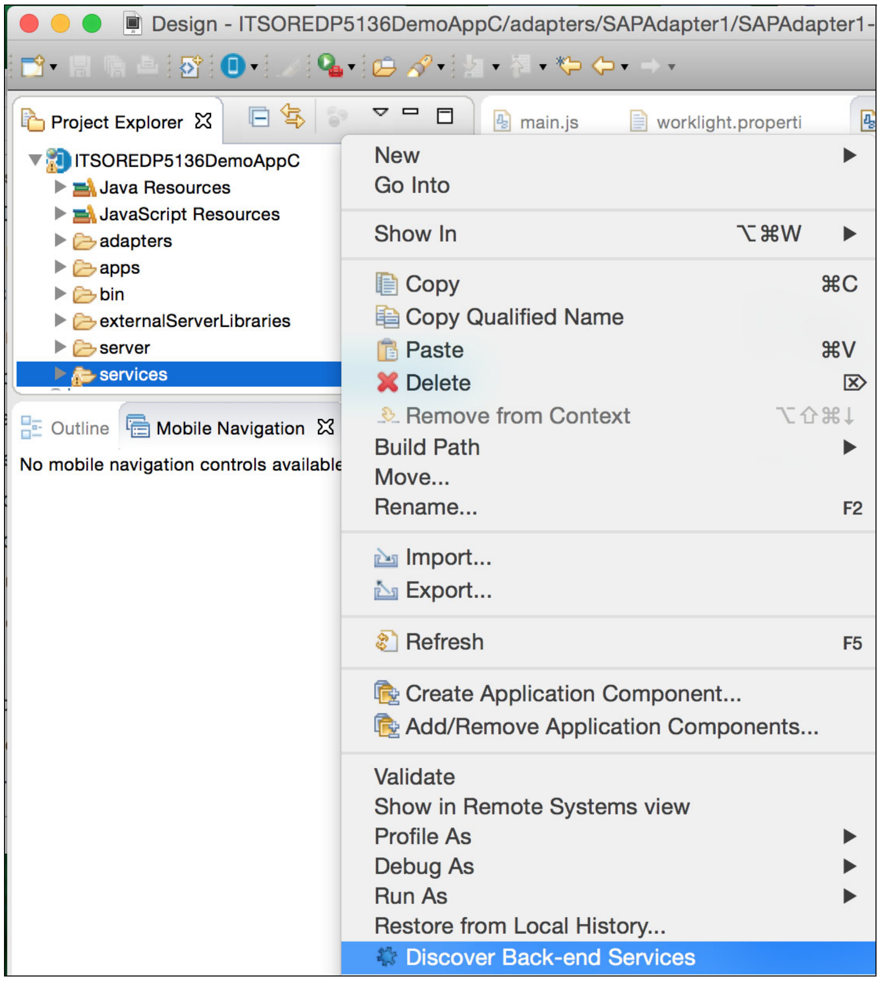
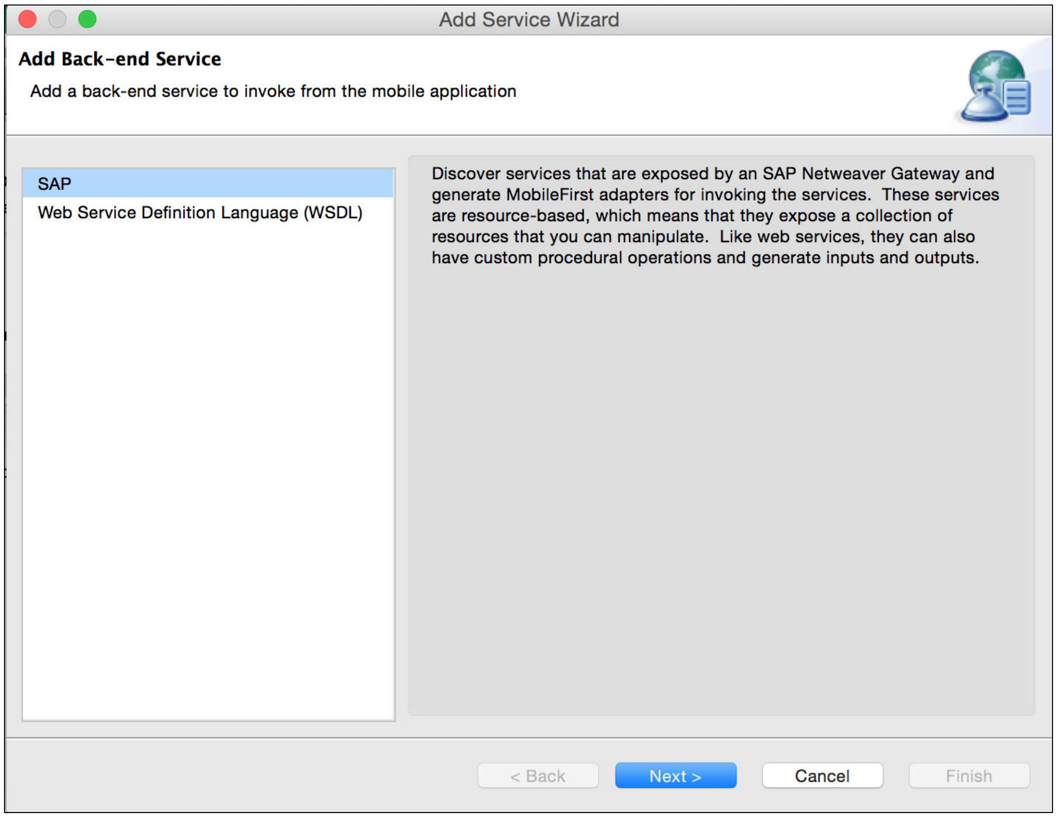


Figure 2-15 Selecting Discover Back-end Services

2. In the Add Service Wizard, select **SAP** as the discovery type and click **Next** (Figure 2-16).


Add Service Wizard
Add Back-end Service
Add a back-end service to invoke from the mobile application
SAP
Web Service Definition Language (WSDL)
< Back
Next >
Cancel
Finish

Discover services that are exposed by an SAP Netweaver Gateway and generate MobileFirst adapters for invoking the services. These services are resource-based, which means that they expose a collection of resources that you can manipulate. Like web services, they can also have custom procedural operations and generate inputs and outputs.

Figure 2-16 Select SAP discovery type

3. Select a suitable SAP connection from the Connection drop-down list (Figure 2-17 on page 49). Select the services, **RMTSAMPLEFLIGHT_2** in this scenario, select the **CarrierCollection** object, and then click **Finish**.

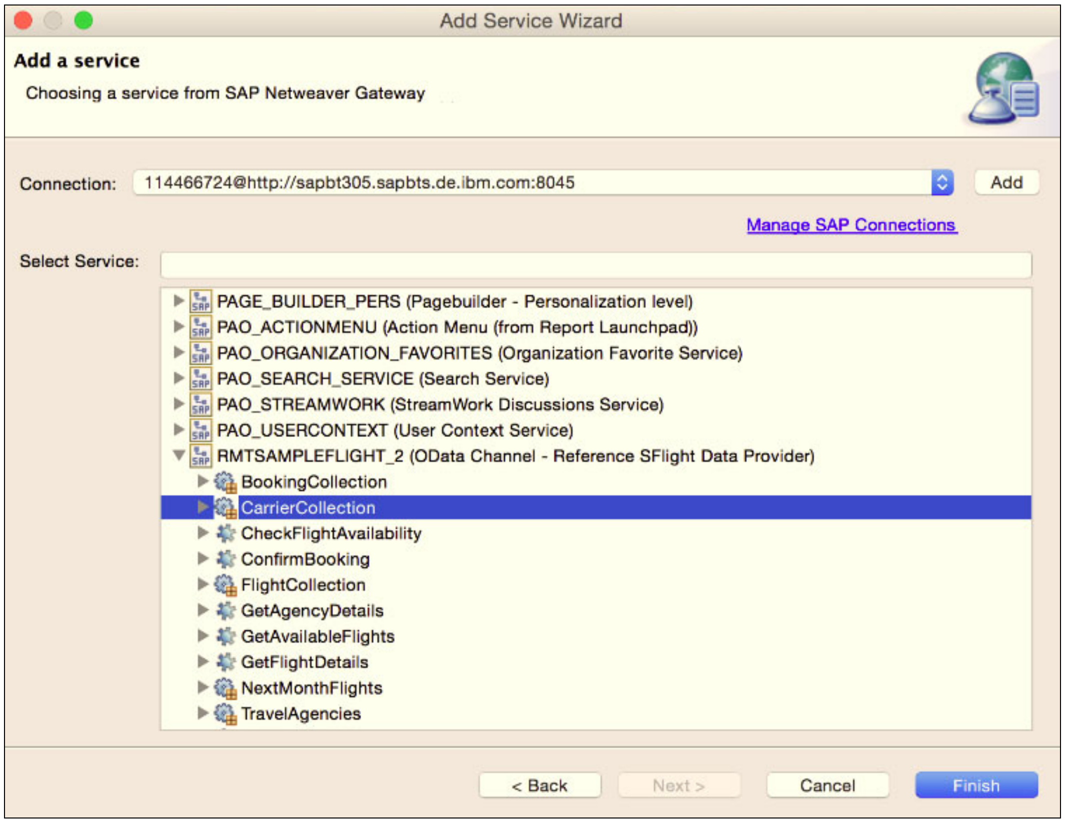

Add Service Wizard
Add a service
Choosing a service from SAP Netweaver Gateway
Connection: 114466724@http://sapbt305.sapbts.de.ibm.com:8045
Add
Manage SAP Connections
Select Service:
PAGE_BUILDER_PERS (Pagebuilder - Personalization level)
PAO_ACTIONMENU (Action Menu (from Report Launchpad))
PAO_ORGANIZATION_FAVORITES (Organization Favorite Service)
PAO_SEARCH_SERVICE (Search Service)
PAO_STREAMWORK (StreamWork Discussions Service)
PAO_USERCONTEXT (User Context Service)
RMTSAMPLEFLIGHT_2 (OData Channel - Reference SFlight Data Provider)
BookingCollection
CarrierCollection
CheckFlightAvailability
ConfirmBooking
FlightCollection
GetAgencyDetails
GetAvailableFlights
GetFlightDetails
NextMonthFlights
TravelAgencies
< Back
Next >
Cancel
Finish

Figure 2-17 Select appropriate SAP NetWeaver Gateway service

4. The discovery wizard generates an adapter instance with all the basic artifacts needed to interact with the SAP NetWeaver Gateway service that are selected for this scenario. Validate that the files are available in the adapter directory (Figure 2-18).

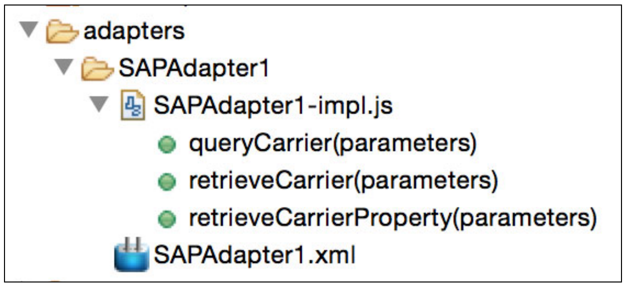


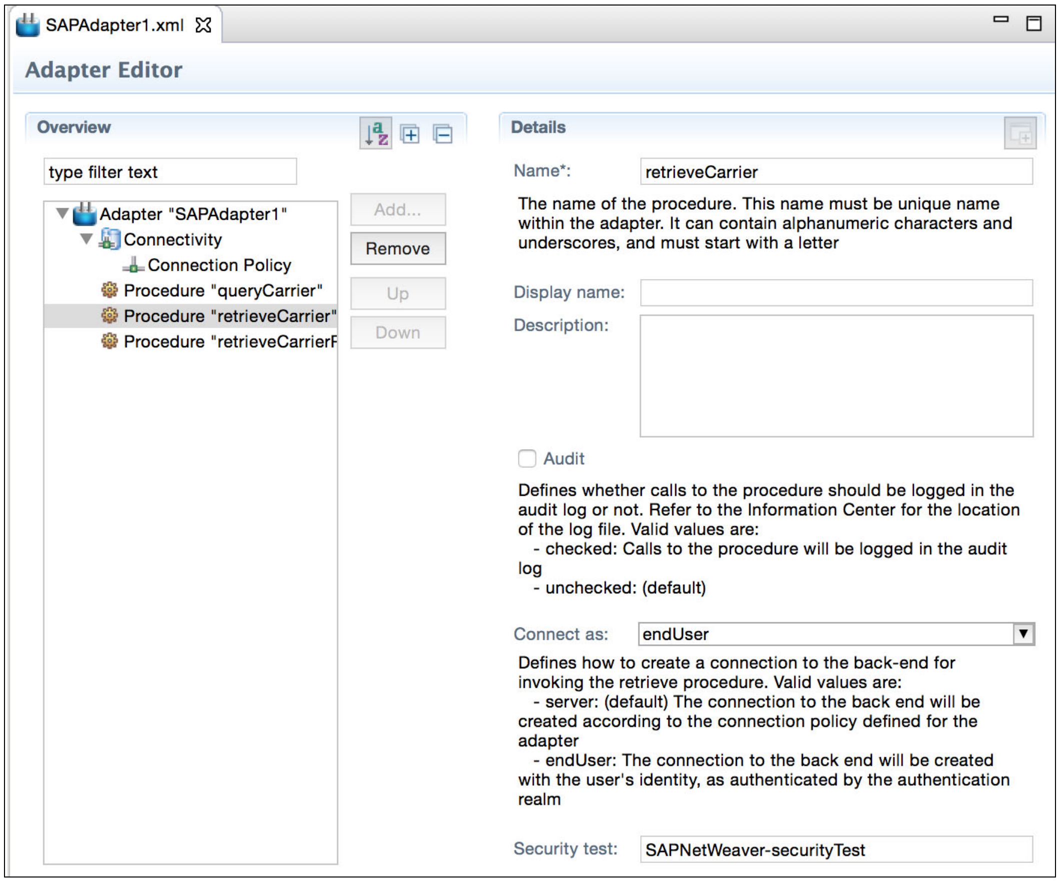
Figure 2-18 Generated adapter artifacts

5. Open the `retrieveCarrier` method and modify the code as shown in Example 2-9.

Example 2-9 Assign response object to secreteBusinessData variable

```
function retrieveCarrier(parameters) {
        var request = {
           CollectionName: "CarrierCollection",
           Keys: [{Name: "AirLineID", value: parameters.keys.AirLineID}, ],
           Select: parameters.select || [],
           Expand: parameters.expand || []
        };
        var response = WL.Server.fetchNWBusinessObject(request)
        return { secretBusinessData: response };
    }
```

6. Open the `SAPAdapter1.xml` file (Figure 2-19). Assign the security test, `SAPNetWeaver-securityTest`, to the `retrieveCarrier` procedure name. Select **endUser** as the connection mechanism (Connect as).



Name*: retrieveCarrier

The name of the procedure. This name must be unique name within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter

Display name:

Description:

Audit

Defines whether calls to the procedure should be logged in the audit log or not. Refer to the Information Center for the location of the log file. Valid values are:
 - checked: Calls to the procedure will be logged in the audit log
 - unchecked: (default)

Connect as: endUser

Defines how to create a connection to the back-end for invoking the retrieve procedure. Valid values are:
 - server: (default) The connection to the back end will be created according to the connection policy defined for the adapter
 - endUser: The connection to the back end will be created with the user's identity, as authenticated by the authentication realm

Security test: SAPNetWeaver-securityTest

Figure 2-19 Assign security test to adapter procedure

7. Save the `SAPAdapter1.xml` file.

Now, a new adapter is available in the project to interact with the SAP NetWeaver Gateway service. The changes in the security configuration of the adapter ensure that it can be used only with a validated security realm.

The `Connect as` parameter defines how to create a connection to the back-end system to invoke the procedure that retrieves the business data. Valid values are as follows:

- `server`: (default) The connection to the back-end system is created according to the connection policy that is defined for the adapter.
- `endUser`: The connection to the back-end system is created with the user's identity, as authenticated by the authentication realm.

In this scenario, the IBM MobileFirst security framework takes the identity of the logged-in user and uses these credentials to access the SAP NetWeaver Gateway service.

2.3.3 Implement the MobileFirst SAP NetWeaver Gateway adapter in the hybrid sample application

The `main.js` JavaScript file contains the application-specific code to call the IBM MobileFirst SAP NetWeaver Gateway adapter.

To add the adapter to the mobile application, complete the following steps:

1. Open the `main.js` file.
2. Add the code snippet, shown in Example 2-10, to call the adapter procedure that returns business data from the SAP back-end system.

Example 2-10 JavaScript code to call adapter

```
function getSecretBusinessData(){
   var invocationData = {
         adapter: "SAPAdapter1",
         procedure: "retrieveCarrier",
         parameters: [{"keys" : {"AirLineID" : "LH"}}]
   };

   WL.Client.invokeProcedure(invocationData, {
      onSuccess: getSecretBusinessData_Callback,
      onFailure: getSecretBusinessData_Callback
   });
}
```

3. The response is handled by a dedicated callback handler method, as shown in Example 2-11.

Example 2-11 JavaScript code to handle response

```
function getSecretBusinessData_Callback(response){
   if(response.invocationResult.isSuccessful) {

      $('#AppHomeResults').html("Secret business data is: <br>" +
JSON.stringify(response.invocationResult.secretBusinessData.AirLineName));

      if(response && (response.invocationResult) &&
(response.invocationResult['WL-Authentication-Success']) &&
(response.invocationResult['WL-Authentication-Success']['SAPNetWeaverRealm']))
      {
         var realm =
response.invocationResult['WL-Authentication-Success']['SAPNetWeaverRealm'];
         $('#AppHomeLoginDetails').html("Logged in as User: " + realm.userId);

      }
   }else
   {
      $('#AppHomeResults').html(JSON.stringify(response));
      }
}
```

4. Save the `main.js` JavaScript file.

To simplify, the input parameter to the SAP NetWeaver Gateway service is hardcoded in this example. In this scenario, the name for the airline code LH, Lufthansa, is always retrieved.

2.3.4 Test the mobile application

The mobile application can be tested with a web browser. The Chrome browser is used in this scenario to open the MobileFirst Console and to start the mobile application.

The mobile application calls the SAP NetWeaver Gateway service, `RMTSAMPLEFLIGHT_2`. This services translates an airline code to the corresponding full airline name.

To test the mobile application, complete the following steps:

1. Build the mobile application and deploy it to the IBM MobileFirst Studio Development Server by using the context menu.
2. Open the MobileFirst Console and select **Preview Common Resource**.
3. Click **Call protected adapter proc** (Figure 2-20).

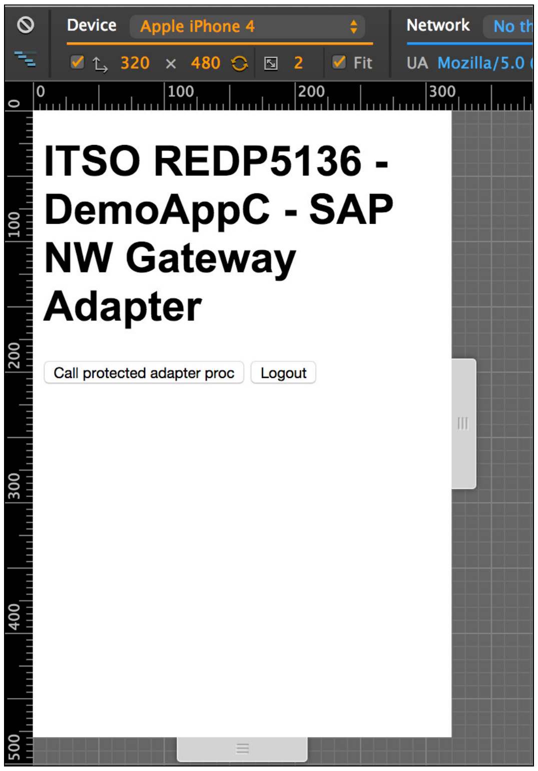
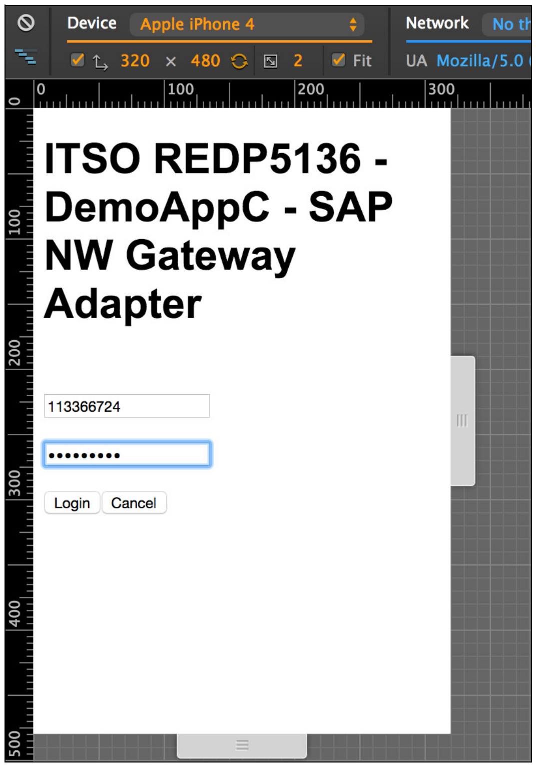

Device
Apple iPhone 4
Network
No th
320 × 480
2
Fit
UA Mozilla/5.0
ITSO REDP5136 - DemoAppC - SAP NW Gateway Adapter
Call protected adapter proc
Logout

Figure 2-20 Call protected resource in the mobile application

4. Enter valid credentials in the login form and click **Login** (Figure 2-21 on page 53).



ITSO REDP5136 - DemoAppC - SAP NW Gateway Adapter

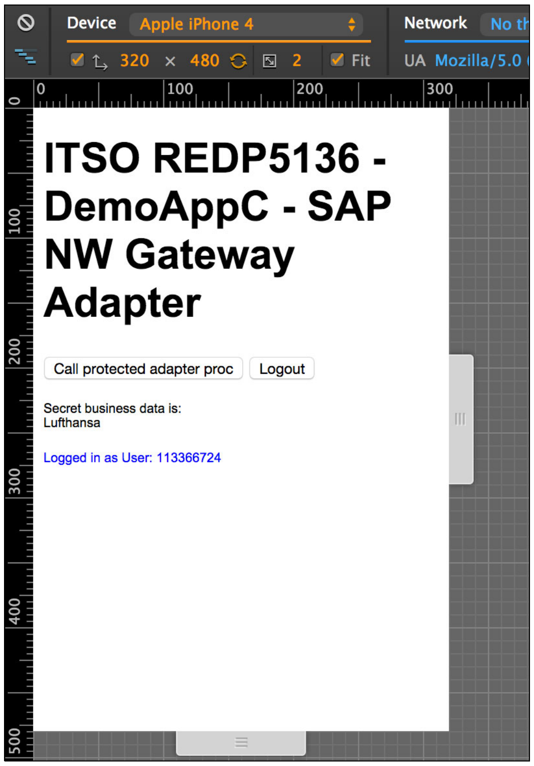
113366724

•••••••••

Login Cancel

Figure 2-21 Enter valid credentials

5. The mobile application shows the secret business data and the logged-in user (Figure 2-22).



ITSO REDP5136 - DemoAppC - SAP NW Gateway Adapter

Call protected adapter proc Logout

Secret business data is:
Lufthansa

Logged in as User: 113366724

Figure 2-22 Results as provided by the SAP NetWeaver Gateway service

3

Offline SAP business data storage and synchronization in mobile scenarios

This chapter describes the offline capabilities of the IBM MobileFirst Platform Foundation v6.3.0. It demonstrates, through a practical example, how a MobileFirst hybrid application can store business data locally in the mobile device; users can work with the data whether the mobile device is connected or not.

This chapter also shows how to synchronize business data between the mobile device and the originating SAP ERP system to build a comprehensive mobile enterprise solution by using the IBM MobileFirst Platform Foundation.

3.1 IBM MobileFirst Platform Foundation offline data capabilities

One of the main usage patterns for an enterprise mobile strategy is workforce enablement or empowerment with the objective of increasing the workforce effectiveness, productivity, and responsiveness. Mobile employees usually require being able to access and process business data from back-end systems in their mobile devices whether network connectivity is available or not.

This requirement implies the following additional requirements:

- Securing the business data on the mobile device.
- Keeping the business data synchronized with the originating system of record.
- Making the business data available in the mobile device also when the device is not connected to the corporate network or when the device is offline.

Having a flexible configurable offline capability is a key requirement for mobile enablement platforms to support the requirements of mobile enterprise applications.

The IBM MobileFirst Platform Foundation includes a feature called JSONStore for data exchange and storage. JSONStore is a lightweight, document-oriented storage system that is included in the product and enables persistent storage of JSON documents on the mobile device. This feature adds the capability to store JSON documents in MobileFirst applications. With it, you can create, read, update, and delete data records from a data source. Each operation is queued when operating offline. When a connection is available, the operation is transferred to the server and each operation is then performed against the source data.

Documents in an application are available in the JSONStore even when the device that is running the application is offline. This persistent, always-available storage can be useful for customers, employees, or partners, to give them access to documents when, for example, there is no network connectivity or access to the enterprise network.

Mobile application developers can use the JSONStore component through a set of well-defined APIs. The following APIs are provided:

- JavaScript API for storing data in hybrid MobileFirst applications.
- Objective-C API for native iOS applications.
- Java API for native Android applications.

An advantage of the JSONStore component is that the developer can focus on the core mobile application rather than developing platform features. The JSONStore provides configurable capabilities such as these:

- Data encryption
- Reliable storage of data
- Implicit tracking of local changes
- Multi user support
- Indexing
- Storage expansion up to the physical device limit

The JSONStore is a file in the mobile device that persists the business data. It is similar to a relational database in database terminology. Within the JSONStore, data is held in collections that represent business data of the same type. The concept is similar to a table in database terminology. The collections hold documents that are the basic building blocks, similar to a record or a row in database terminology.

The JSONStore API provides enhanced and robust functions to manage the collections and documents within the store. Special focus is given to the search capabilities that allow a mobile developer to find business data fast.

Figure 3-1 shows a MobileFirst mobile application that uses the MobileFirst JSONStore API to interact with the embedded JSONStore database.

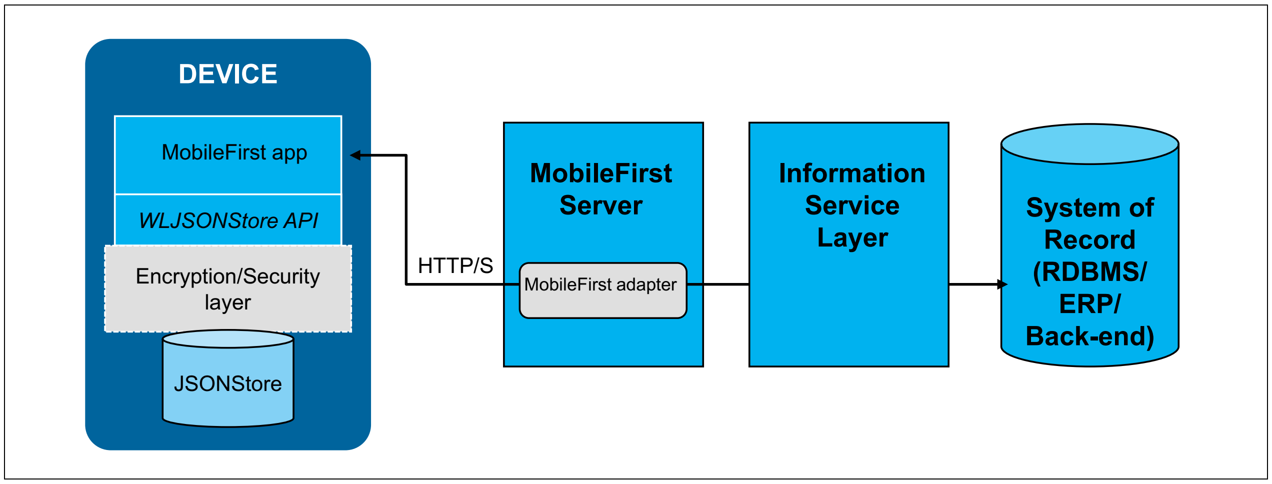

DEVICE
MobileFirst app
WLJSONStore API
Encryption/Security layer
JSONStore
HTTP/S
MobileFirst Server
MobileFirst adapter
Information Service Layer
System of Record (RDBMS/ ERP/ Back-end)

Figure 3-1 JSONStore core components and interaction

The database can be encrypted. The JSONStore API supports full synchronization with the back-end system through a MobileFirst adapter. The data that is changed in the back end is synchronized with the JSONStore through adapter calls. Changes on the data in the JSONStore while the device is offline are updated to the back end when a connection becomes available through MobileFirst adapter calls.

Besides providing embedded encryption and security layer, the JSONStore integration with the IBM MobileFirst Platform adapter framework is an advantage for mobile developers who build enterprise mobile apps. With this capability, developers can easily define relationships between JSONStore data objects that are stored in collections and MobileFirst adapter functions that read and write to the connected back-end systems.

3.2 Developing a mobile application with offline capabilities

This section describes the steps to develop a mobile application that includes the IBM MobileFirst Platform offline capabilities. The application uses the JSONStore to keep a collection of business data on the mobile device. Additionally, it leverages the integration with the MobileFirst adapter framework to interact with an IBM Cast Iron orchestration. The IBM Cast Iron orchestration provides, to the consumers, a REST API while interacting with the underlying SAP ERP system by using SAP remote function calls.

The business scenario demonstrates the replication of customer data to the mobile device. Mobile users can pull the customer records from the SAP ERP system, make changes, and push the changed and new records back to the SAP ERP system.

3.2.1 IBM Cast Iron orchestration overview

For the overall understanding of this example, understanding the functionality of the IBM Cast Iron orchestration, which provides the integration with the SAP ERP system, is important. The IBM Cast Iron orchestration calls three SAP OLAP Business Application Programming Interface (BAPI) function modules:

- BAPI_FLCUST_CREATEFROMDATA

  This function module creates a new flight customer record in the SAP system.
- BAPI_FLCUST_CHANGE

  This function module changes an existing flight customer record in the SAP system.
- BAPI_FLCUST_GETLIST

  This function module returns a list of all existing flight customer records in the SAP system.

The IBM Cast Iron orchestration is built by using the IBM Cast Iron Studio, which is a graphical editor that helps integration developers to visualize the integration component, as shown in Figure 3-2.

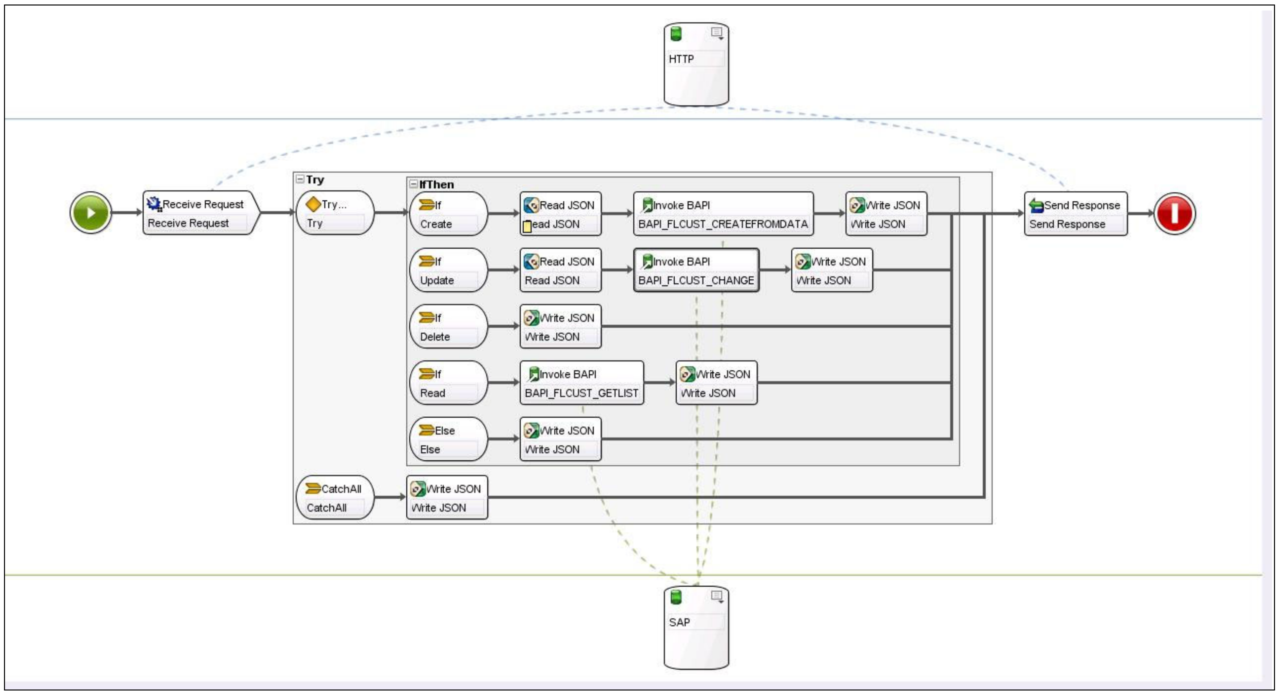


Figure 3-2 IBM CastIron orchestration FlightCustomer API

The orchestration provides an HTTP endpoint to calling clients and it is secured by basic authentication. Incoming HTTP requests that fit the configured listening context URI are received by the `Receive Request` activity and the relevant input parameters are extracted as shown in Figure 3-3 on page 59.

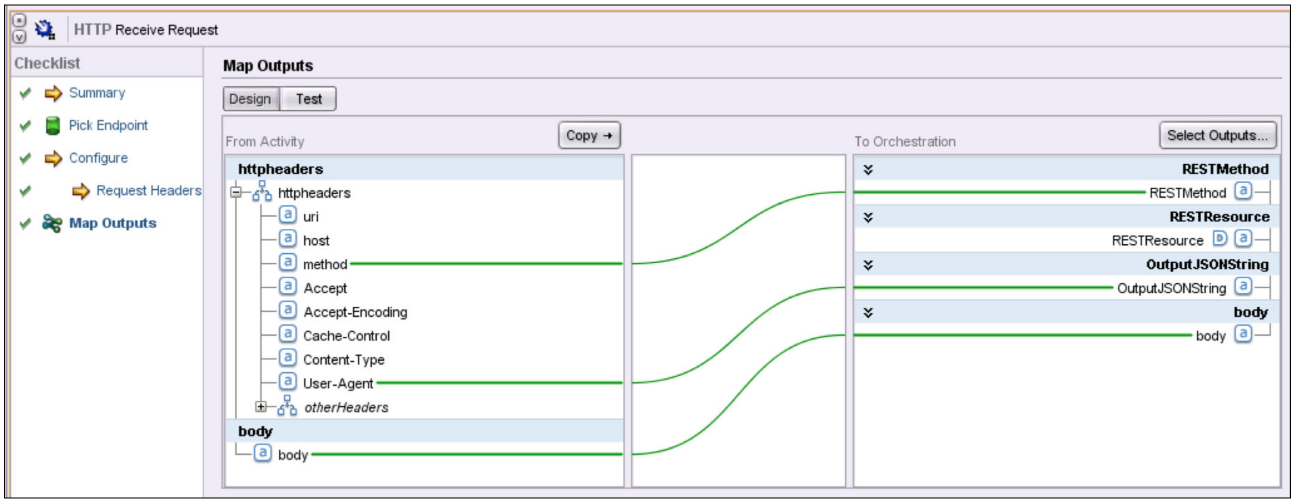
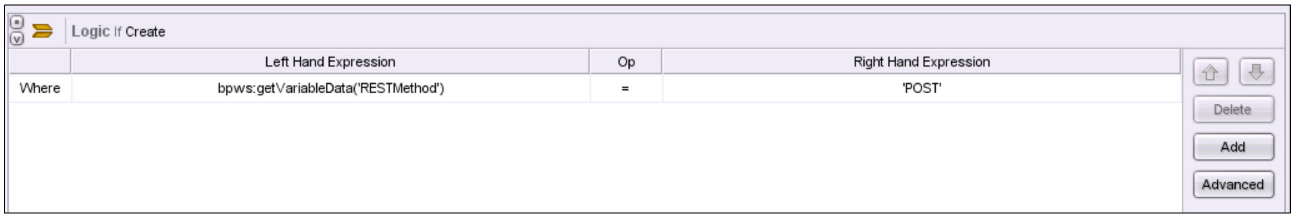


Figure 3-3 Extraction of HTTP input parameters

Based on the HTTP method of the incoming call, the orchestration flow decides which path of the `IF` statement construct to follow. The decision criteria is configured as shown in Figure 3-4.



	Left Hand Expression	Op	Right Hand Expression
Where	bpws:getVariableData('RESTMethod')	=	'POST'

Figure 3-4 Deciding which path to follow based on the HTTP method

The `ReadJSON` activity is responsible to parse the input body and to make the received JSON payload accessible within the IBM Cast Iron orchestration as shown in Figure 3-5.

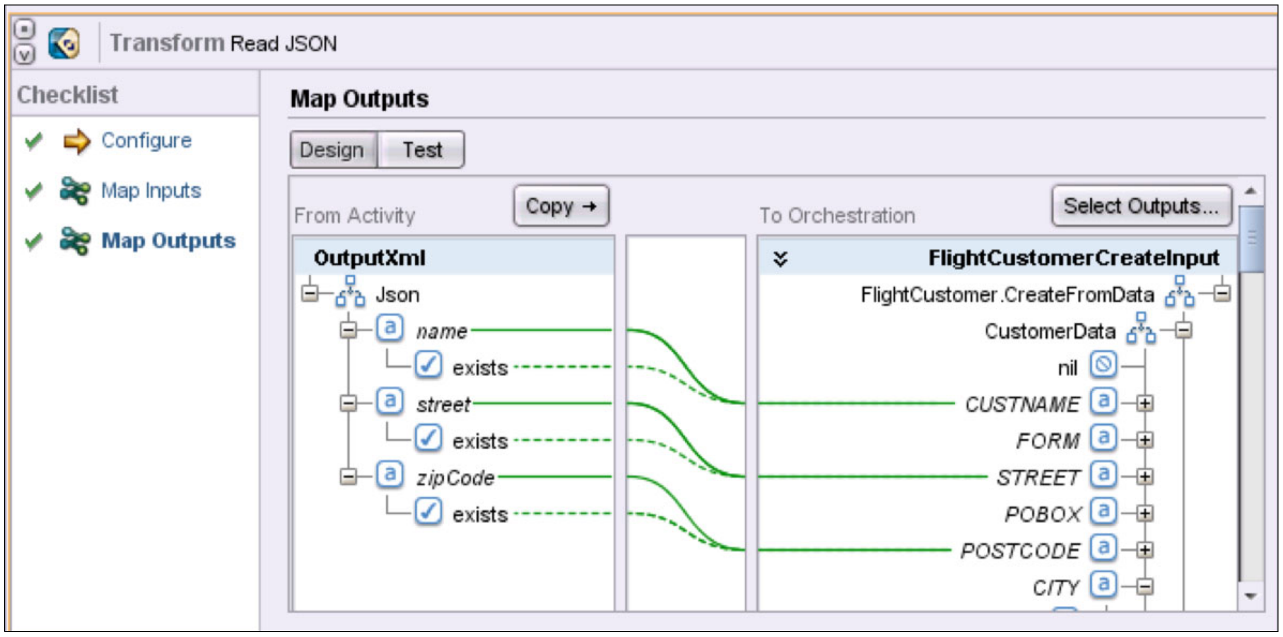
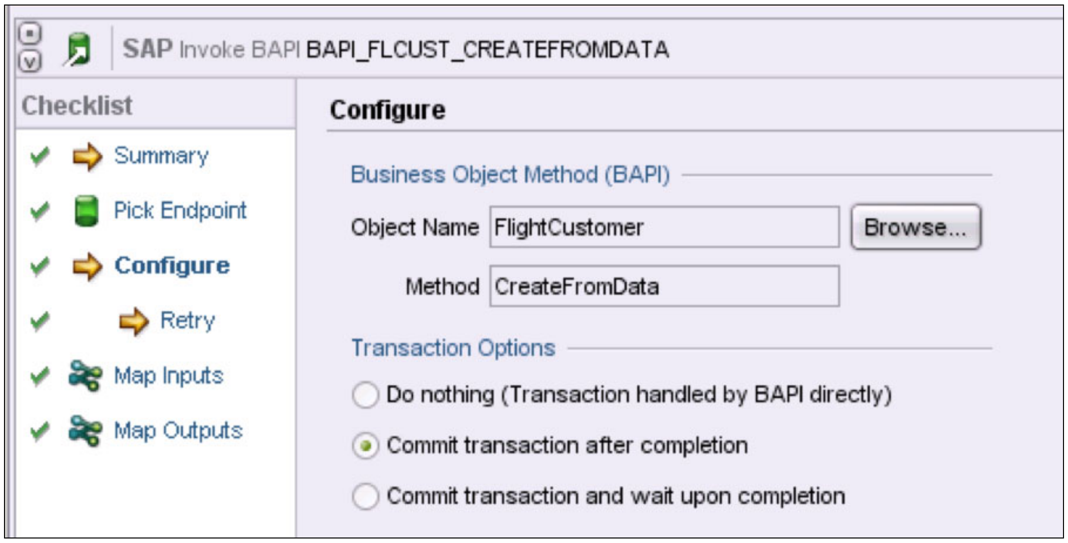


Figure 3-5 Parsing of the HTTP body element to receive the JSON payload

The `Invoke BAPI` activity is responsible for triggering the SAP call, passing the received input values, and receiving the results from the SAP system. In this scenario, an important steps is to configure this activity to trigger an explicit commit, otherwise the data is not persisted with the SAP system. The configuration details differ for each SAP function module and they must be discussed with the appropriate SAP administration team. The `Invoke BAPI` activity has various settings to accommodate nearly any requirement, as shown in Figure 3-6.


SAP Invoke BAPI BAPI_FLCUST_CREATEFROMDATA

Checklist
- ✓ Summary
- ✓ Pick Endpoint
- ✓ Configure
- ✓ Retry
- ✓ Map Inputs
- ✓ Map Outputs

Configure

Business Object Method (BAPI)

Object Name: FlightCustomer Browse...

Method: CreateFromData

Transaction Options
- ○ Do nothing (Transaction handled by BAPI directly)
- [✓ Commit transaction after completion]
- ○ Commit transaction and wait upon completion

Figure 3-6 Configuring the SAP interaction type to force a transaction commit

The received SAP business data is transformed to JSON and returned by the `Send Response` activity.

3.2.2 Create a hybrid application skeleton using IBM MobileFirst Studio

Create a hybrid application by using the corresponding Eclipse wizard within the MobileFirst Studio.

1. Select **MobileFirst Project** (Figure 3-7).

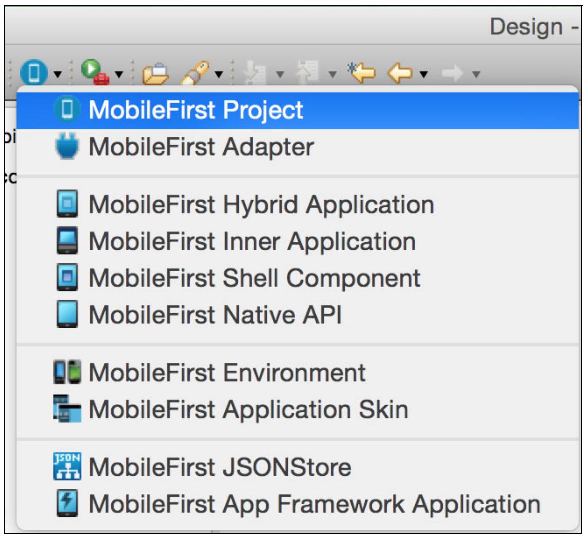
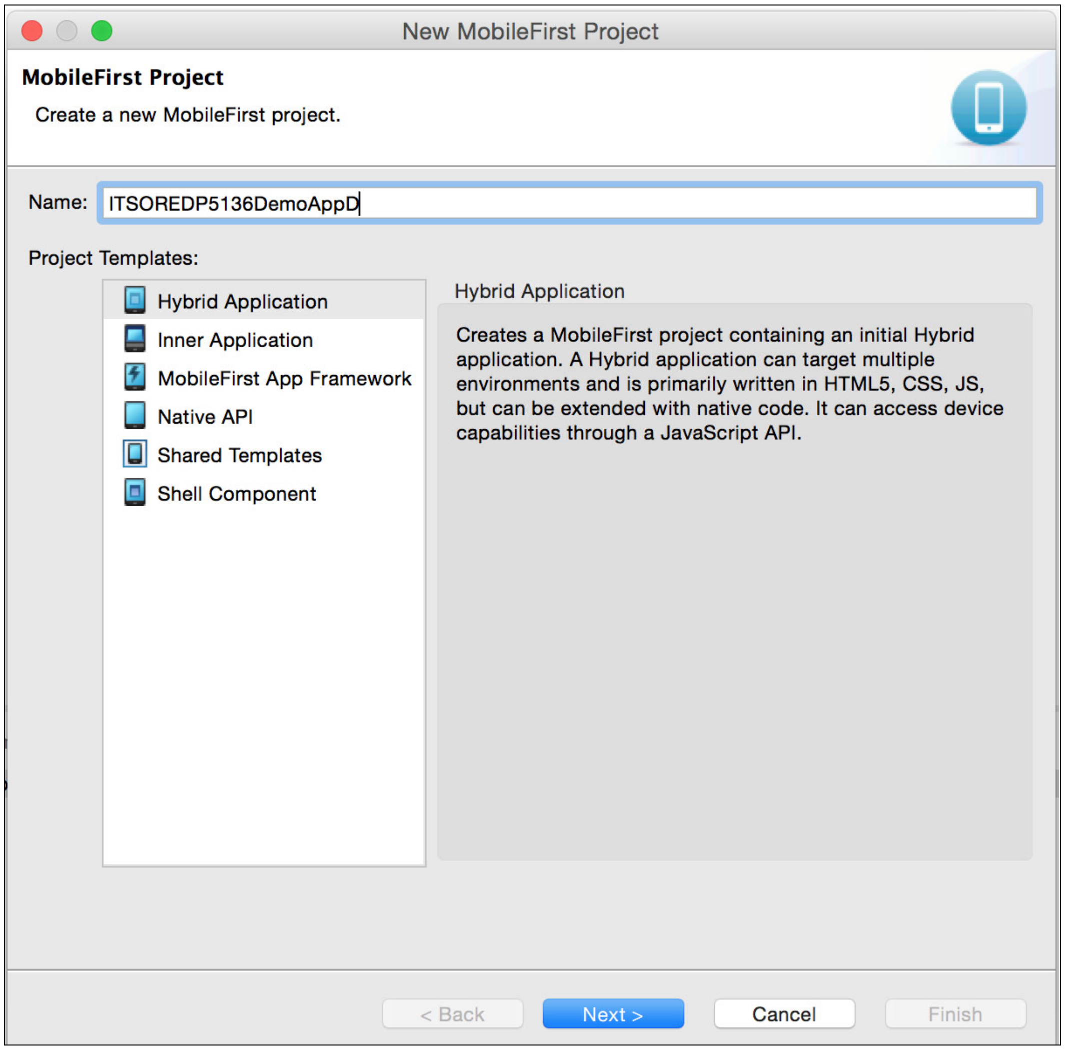

Design
MobileFirst Project
MobileFirst Adapter
MobileFirst Hybrid Application
MobileFirst Inner Application
MobileFirst Shell Component
MobileFirst Native API
MobileFirst Environment
MobileFirst Application Skin
MobileFirst JSONStore
MobileFirst App Framework Application

Figure 3-7 Create a MobileFirst project by using Eclipse wizard

2. Enter a suitable project name, for example `ITSOREDP5136DemoAppD`, and select the **Hybrid Application** template (Figure 3-8).



MobileFirst Project

Create a new MobileFirst project.

Name: ITSOREDP5136DemoAppD

Project Templates:

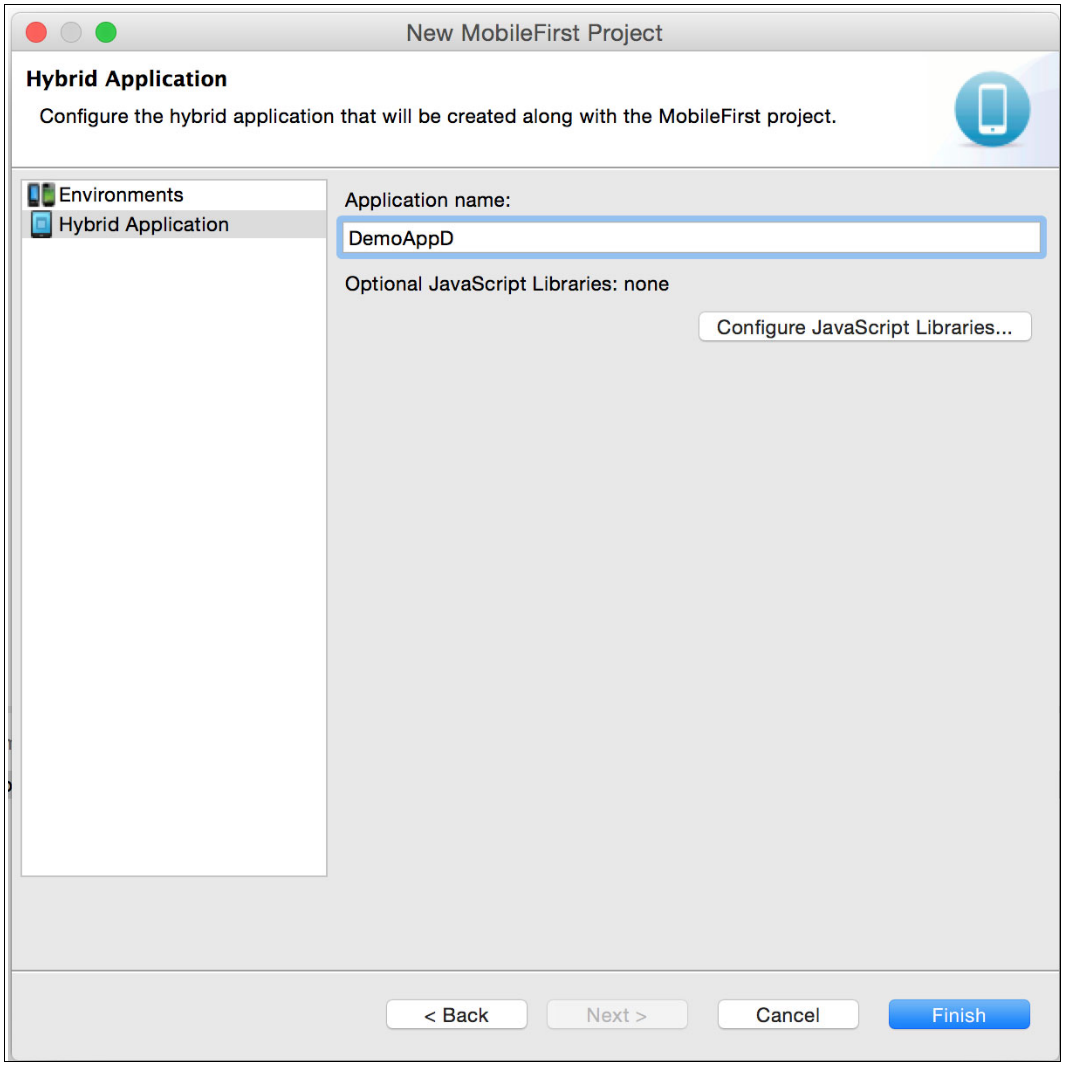
- Hybrid Application
- Inner Application
- MobileFirst App Framework
- Native API
- Shared Templates
- Shell Component

Hybrid Application

Creates a MobileFirst project containing an initial Hybrid application. A Hybrid application can target multiple environments and is primarily written in HTML5, CSS, JS, but can be extended with native code. It can access device capabilities through a JavaScript API.

Figure 3-8 Select Hybrid Application type

3. Enter a suitable application name, for example `DemoAppD`, and click **Finish** (Figure 3-9).


New MobileFirst Project

Hybrid Application

Configure the hybrid application that will be created along with the MobileFirst project.

Application name: DemoAppD

Optional JavaScript Libraries: none

Figure 3-9 Enter application name

Now, you have an empty mobile hybrid application project. It is extended, step-by-step, in the following sections, demonstrating the IBM MobileFirst Platform offline capability, which enables business data synchronization with an SAP Enterprise Resource Planning system.

3.2.3 Create a MobileFirst Cast Iron adapter in the hybrid application project

To create a MobileFirst Cast Iron adapter in the MobileFirst project (created in 3.2.2, "Create a hybrid application skeleton using IBM MobileFirst Studio" on page 60), complete the following steps:

1. Click the **Create a MobileFirst Artifact** icon in the toolbar and select **MobileFirst Adapter** (Figure 3-10).

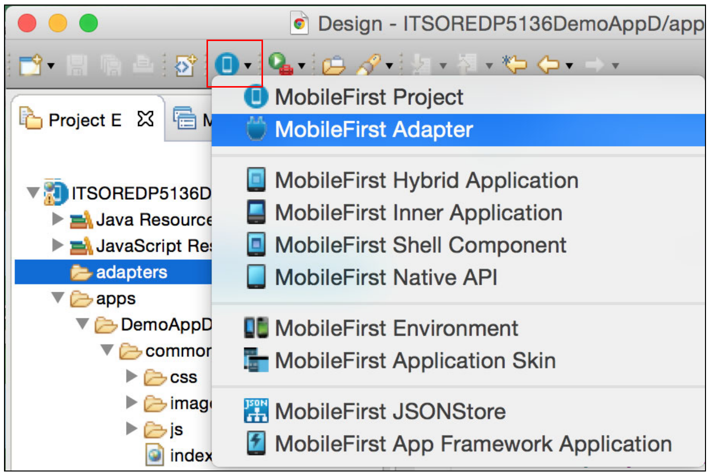
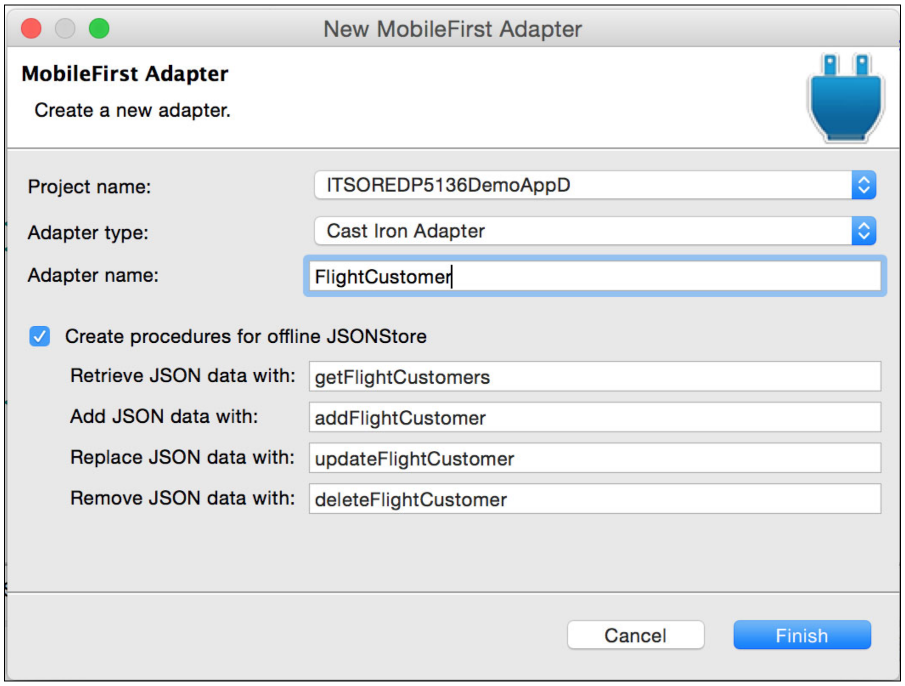


Figure 3-10 Create a MobileFirst adapter using the Eclipse wizard

2. Select adapter type **Cast Iron Adapter**, enter a meaningful adapter name, for example `FlightCustomer`, and define the names of the JavasScript procedures for the JSONStore linkage as shown in Figure 3-11. Then, click **Finish**.


New MobileFirst Adapter

MobileFirst Adapter

Create a new adapter.

Project name:	ITSOREDP5136DemoAppD
Adapter type:	Cast Iron Adapter
Adapter name:	FlightCustomer

✓ Create procedures for offline JSONStore

Retrieve JSON data with:	getFlightCustomers
Add JSON data with:	addFlightCustomer
Replace JSON data with:	updateFlightCustomer
Remove JSON data with:	deleteFlightCustomer

Figure 3-11 Define Adapter type and JSONStore JavaScript procedures

Modify the adapter XML file

The adapter XML file defines technical details of the generated MobileFirst Cast Iron adapter. To modify the adapter XML file for this scenario, complete the following steps:

1. Open the `FlightCustomer.xml` file.
2. Modify the `connectionPolicy` XML tag to include the entries shown in Example 3-1.

Example 3-1 MobileFirst Adapter configuration

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
         <protocol>https</protocol>
         <domain>eval-provide.castiron.com</domain>
         <port>443</port>
         <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
         <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
         <authentication>
            <basic />
            <serverIdentity>
               <username>UserID</username>
               <password>password</password>
            </serverIdentity>
         </authentication>
         <maxConcurrentConnectionsPerNode>4</maxConcurrentConnectionsPerNode>
      </connectionPolicy>
```

3. Add valid credentials in the `username` and `password` tags; the credentials must be accepted by the corresponding IBM Cast Iron orchestration.
4. Modify the generated adapter procedure definition to use the `connectAs` attribute server as shown Example 3-2.

Example 3-2 Adapter procedure connection option

```
   <procedure name="getFlightCustomers" connectAs="server"> </procedure>
   <procedure name="addFlightCustomer" connectAs="server"> </procedure>
   <procedure name="updateFlightCustomer" connectAs="server"> </procedure>
   <procedure name="deleteFlightCustomer" connectAs="server"> </procedure>
```

5. **Save** the adapter XML file.

Modify the adapter JavaScript file

The adapter JavaScript file contains the JavaScript logic that represents the IBM MobileFirst adapter. In this scenario, it includes four functions that represent the interactions with the SAP system of record when an object is created, changed, deleted, or retrieved. The generated method stubs are empty and must be filled as follows:

1. Open the `FlightCustomer-impl.js` JavaScript file.
2. Modify the `addFlightCustomer` JavaScript function (Example 3-3) to implement the functionality that adds a customer record to the back-end SAP system.

Example 3-3 Adapter logic to create a new flight customer record in the SAP system

```
function addFlightCustomer(param1) {

   WL.Logger.info('Adapter: FlightCustomer, procedure: addFlightCustomer
called.');

   var input = {
```

```
            method  : 'post',
            appName : 'ITSOREDP5136DemoAppD',
            requestType: 'http',
            path : '/env/Development/sapapi/flightcustomers',
            returnedContentType : 'json',
            body:
            {
               content: param1.toString(),
               contentType: 'application/json'
            }
      };
      WL.Logger.info('Sending data: ' + JSON.stringify(input));

      var response = WL.Server.invokeCastIron(input);

      WL.Logger.info('Got data from JSONStore to ADD: ' +
   JSON.stringify(response));

      return;
   }
```

3. Modify the `updateFlightCustomer` JavaScript function (Example 3-4) to update a customer record in the SAP back-end system.

Example 3-4 Adapter logic to update existing flight customer record in the SAP system

```
function updateFlightCustomer(param1) {

   WL.Logger.info('Adapter: FlightCustomer, procedure: updateFlightCustomer
called.');

   var input = {
         method  : 'put',
         appName : 'ITSOREDP5136DemoAppD',
         requestType: 'http',
         path : '/env/Development/sapapi/flightcustomers',
         returnedContentType : 'json',
         body:
         {
            content: param1.toString(),
            contentType: 'application/json'
         }
   };
   WL.Logger.info('Sending data: ' + JSON.stringify(input));

   var response = WL.Server.invokeCastIron(input);

   WL.Logger.info('Got data from JSONStore to REPLACE: ' +
JSON.stringify(response));

   return;
}
```

4. Modify the `getFlightCustomers` JavaScript function (Example 3-5) to read all customer records from the SAP back-end system.

Example 3-5 Adapter logic to retrieve flight customer list from the SAP system

```
function getFlightCustomers() {

   WL.Logger.info('Adapter: FlightCustomer, procedure: getFlightCustomers 
called.');

   var input = {
         method  : 'get',
         appName : 'ITSOREDP5136DemoAppD',
         requestType: 'http',
         path : '/env/Development/sapapi/flightcustomers',
         returnedContentType : 'json'
   };
   WL.Logger.info('Sending data: ' + JSON.stringify(input));

   var response = WL.Server.invokeCastIron(input);

   WL.Logger.info('Receiving data: ' + 
JSON.stringify(response.flightcustomerList));

   return { flightcustomerlist: response.flightcustomerList };

}
```

5. Modify the `deleteFlightCustomer` JavaScript function with the JavaScript code that writes the input to the log file as shown in Example 3-6.

Note: The SAP Flight Data application has no delete function. In this example, the Delete stub method created by MobileFirst Studio was modified to log the changed records so you can verify what was changed.

Example 3-6 Adapter logic to delete flight customer record in the SAP system

```
function deleteFlightCustomer(param1) {

   WL.Logger.info('Adapter: FlightCustomer, procedure: deleteFlightCustomer 
called.');
   WL.Logger.info('Got data from JSONStore to REMOVE: ' + param1);

   return;
}
```

6. Save the `FlightCustomer-impl.js` file.

Note: The `deleteFlightCustomer` method does not call the SAP ERP system because this SAP business object does not provide an appropriate BAPI for this operation.

The adapter is now ready to interact with the IBM Cast Iron orchestration to exchange flight customer business data and to provide this data to the JSONStore component.

3.2.4 Add custom stylesheet content

The cascading style sheet of the mobile application is not relevant for the offline capability that this example demonstrates, but it makes the example mobile app look more appealing.

Complete the following steps:

1. Open the `main.css` file in the `common/css` folder.
2. Add the code snippet, shown in Example 3-7, to provide the style sheet settings.

Example 3-7 Implement stylesheet for sample application

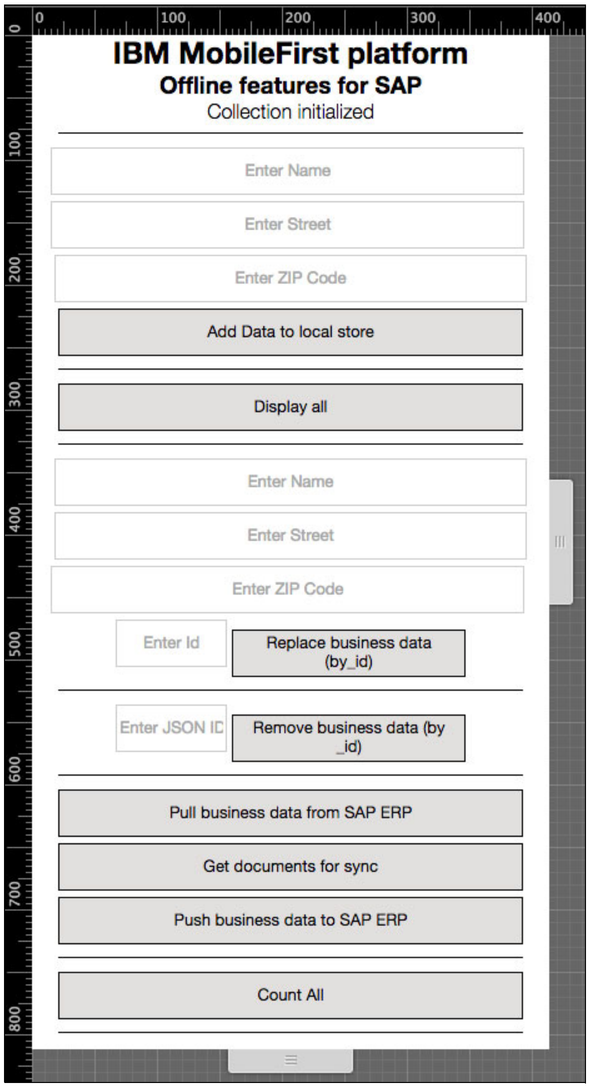
```
a, abbr, address, article, aside, audio, b, blockquote, body, canvas, caption, cite,
code, dd, del, details, dfn, dialog, div, dl, dt, em, fieldset, figcaption, figure,
footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, html, i, iframe, img, ins, kbd,
label, legend, li, mark, menu, nav, object, ol, p, pre, q, samp, section, small, span,
strong, sub, summary, sup, table, tbody, td, tfoot, th, thead, time, tr, ul, var, video
{
   margin: 0;
   padding: 0;
}

body {
   font-family: "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue",
Helvetica, Arial, "Lucida Grande", sans-serif;
   font-weight: 300;
   text-align: center;
   max-width: 30em;
}

button {
   width: 90%;
   height: 3em;
   font-size: 0.8em;
   margin: 0.2em 0 0.2em 0;
   background: rgb(226, 223, 223);
   border-color: black;
   border-width: 0.1em;
}

input {
   width: 86%;
   height: 6%;
   font-size: 0.8em;
   margin: 0.2em 0 0.2em 0;
   text-align: center;
   height: 2.5em;
}

table {
   margin-top: 0.2em;
   margin-left: auto;
   margin-right: auto;
   text-align: right;
   border-collapse: collapse;
   background-color: Gainsboro;
   padding: 4px;
   border: 1px solid black;
}
```

```
td {
   padding: 3px;
   margin-bottom: 30px;
   border: 1px solid black;
}

tr {
   border: 1px solid black;
}

.button-line-2 {
   width: 45%;
   font-size: 0.8em;
   text-align: center;
   display: inline-block;
}

.button-line-3 {
   width: 29%;
   font-size: 0.7em;
   text-align: center;
   display: inline-block;
}

.button-line-4 {
   width: 29%;
   font-size: 0.7em;
   text-align: center;
   display: inline-block;
}

.input-line-1 {
   width: 90%;
   font-size: 0.8em;
   text-align: center;
   display: inline-block;
}

.input-line-2 {
   width: 45%;
   font-size: 0.8em;
   text-align: center;
   display: inline-block;
}

.input-line-3 {
   width: 30%;
   font-size: 0.8em;
   text-align: center;
   display: inline-block;
}

.input-line-4 {
   width: 20%;
   font-size: 0.8em;
   text-align: center;
   display: inline-block;
}

hr {
  border: 0;
  width: 90%;
  border-top: 0.1em solid black;
  height: 0;
  background: black;
}
```

3. Save the file `main.css` file.

3.2.5 Design the user interface

The user interface of the generated mobile hybrid application is included in the `index.html` file. In this scenario, a screen is displayed that has various buttons and input fields (Figure 3-12).



IBM MobileFirst platform

Offline features for SAP

Collection initialized

Figure 3-12 Sample application home screen

The mobile application screen is divided into seven areas that are separated by black horizontal lines. The purpose of each area is as follows:

- Area 1

  The top area displays a static header and a dynamic status line.

- Area 2

  The first input fields are used when a new local flight customer record must be created. The corresponding button triggers a specific action. The results area is displayed in the status line within the first area.

- Area 3

  The `Display all` button populates the status line at the top with the content of the JSONStore documents for this specific flight customer collection.

- Area 4

  This area is used to change existing records. For this function to work, a valid document ID must be entered.

- Area 5

  This area takes a valid document ID and removes the corresponding document from the local store.

- Area 6

  These three buttons control the interaction with the MobileFirst adapter framework and the underlying SAP system:

  - The `Pull business data from SAP ERP` button removes all existing local content before it triggers a complete reload from the SAP ERP system.
  - The `Get documents for sync` button displays all local documents that are marked as having local changes and that will be transferred to the system of record when the next push is triggered.
  - The `Push business data to SAP ERP` button initiates a push of all documents to the SAP ERP system. The documents included are the newly created, changed, and deleted documents.

- Area 7

  The last area contains one button that counts all flight customer records in the local store and displays the result in the status line at the top.

The `index.html` file with the mobile application logic is shown in Example 3-8.

Example 3-8 Implementation of the index.html file

```
<!DOCTYPE HTML>
<html>
    <head>
      <meta charset="UTF-8">
      <title>Offline features for SAP</title>
      <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0, user-scalable=0">
      <!--
         <link rel="shortcut icon" href="images/favicon.png">
         <link rel="apple-touch-icon" href="images/apple-touch-icon.png">
      -->
      <link rel="stylesheet" href="css/main.css">
      <script>window.$ = window.jQuery = WLJQ;</script>
    </head>
      <body id="content" style="display: none">

      <h2>IBM MobileFirst platform</h2>
      <h3>Offline features for SAP</h3>
      <!--application UI goes here-->

      <div id='status-field'>Offline data store not initialized</div>
      <hr>

      <input id='add-name' class='input-line-1' type='text' placeholder='Enter
Name'/><br>
      <input id='add-street' class='input-line-1' type='text' placeholder='Enter
Street'/><br>
      <input id='add-zipCode' class='input-line-1' type='text' placeholder='Enter
ZIP Code'/>

      <button id='add-data' class='button-line-1'>Add Data to local store</button>

      <hr>
      <button id='find-all' class='button-line-1'>Display all</button>
      <hr>

      <input id='replace-name' class='input-line-1' type='text' placeholder='Enter
Name'/>
      <input id='replace-street' class='input-line-1' type='text'
placeholder='Enter Street'/>
      <input id='replace-zipCode' class='input-line-1' type='text'
placeholder='Enter ZIP Code'/><br>

      <input id='replace-id' class='input-line-4' type='text' placeholder='Enter
Id'/>
      <button id='replace' class='button-line-2'>Replace business data
(by_id)</button>

      <hr>

      <input id='remove-id' class='input-line-4' type='text' placeholder='Enter
JSON ID'/>
      <button id='remove-id-btn' class='button-line-2'>Remove business data (by
_id)</button>

      <hr>

      <button id='load' class='button-line-1'>Pull business data from SAP
ERP</button>
      <button id='get-push-required' class='button-line-1'>Get documents for
sync</button>
      <button id='push' class='button-line-1'>Push business data to SAP
ERP</button>

      <hr>

      <button id='count' class='button-line-1'>Count All</button>
      <hr>
```

```
        <script src="js/initOptions.js"></script>
        <script src="js/main.js"></script>
        <script src="js/messages.js"></script>
    </body>
</html>
```

3.2.6 Develop the mobile application interaction

The interaction aspects of the mobile application, such as the application response when users tap screen buttons, are included in the `main.js` file. This file is a key component of an IBM MobileFirst hybrid application and it is called when the application is initialized. Although encapsulating the custom JavaScript code in a separate file is possible, to simplify this example the code is included in the central `wlCommonInit` method in the `main.js` file.

The jQuery capabilities are used in this method to trigger actions and to manipulate the HTML Document Object Model (DOM). The code uses the JSONStore JavaScript API to interact with the local JSONStore capability. For education purposes, review all code entries that start with the `WL.JSONStore` prefix because this is that main class for the IBM MobileFirst Platform JSONStore API.

Example 3-9 shows the complete listing for the `main.js` file code.

Example 3-9 wlCommonInit implementation

```
function wlCommonInit(){
    /*
     * Use of WL.Client.connect() API before any connectivity to a Worklight Server is required.
     * This API should be called only once, before any other WL.Client methods that communicate
with the Worklight Server.
     * Don't forget to specify and implement onSuccess and onFailure callback functions for
WL.Client.connect(), e.g:
     *
     *    WL.Client.connect({
     *     onSuccess: onConnectSuccess,
     *     onFailure: onConnectFailure
     *    });
     *
     */

    // Common initialization code goes here
    (function (WL, jQuery, lodash) {

    'use strict';

    //Dependencies
    var $ = jQuery,
        _ = lodash;

    //CONSTANTS
    var FLIGHTCUSTOMER_COLLECTION_NAME = 'flightcustomer',
        KEY_VALUE_COLLECTION_NAME = 'keyvalue',
        INIT_FIRST_MSG = 'PERSISTENT_STORE_NOT_OPEN',
        NAME_FIELD_EMPTY_MSG = 'Name field is empty',
        EMPTY_TABLE_MSG = 'No documents found',
        INIT_MSG = 'Collection initialized',
        ADD_MSG = 'Data added to the collection',
        REPLACE_MSG = 'Document replaced succesfully, call find.',
        REMOVE_MSG = 'Documents removed: ',
        COUNT_MSG = 'Documents in the collection: ',
        REMOVE_COLLECTION_MSG = 'Removed all data in the collection',
        LOAD_MSG = 'New documents loaded from adapter: ',
        PUSH_MSG_FAILED = 'Could not push some docs, res: ',
        PUSH_MSG = 'Push finished',
        PASS_CHANGED_MSG = 'Password changed succesfully',
        COUNT_QUERY_ERROR_MSG = 'FIND_BY_QUERY_EXPECTED_A_STRING',
        COUNT_QUERY_MSG = "Documents in the collection with name = ";

    //Log messages to the console and status field
    var _logMessage = function (msg, id) {
        //Get reference to the status field
        var status = _.isUndefined(id) ? $('div#status-field') : $(id);

        //Put message in the status div
        status.text(msg);

        //Log message to the console
        WL.Logger.info(msg);
    };

    // query document which has to be replaced
    var originaldocument = null;

    //Show JSONStore document in a table
    var _showTable = function (arr) {

        if (_.isArray(arr) && arr.length < 1) {
            return _logMessage(EMPTY_TABLE_MSG);
        }
        //Log to the console
        WL.Logger.ctx({stringify: true, pretty: true}).info(arr);

        var
            //Get reference to the status field
            status = $('div#status-field'),

            //Table HTML template
            table = [
                '<table id="flightcustomer_table" >',
                    '<tr>',
                        '<td><b>_id</b></td>',
                        '<td><b>CustomerNr</b></td>',
                        '<td><b>Name</b></td>',
                        '<td><b>Street</b></td>',
                        '<td><b>ZIP</b></td>',
                    '</tr>',
                    '<% _.each(flightcustomerlist, function(flightcustomer) { %>',
                        '<tr>',
                            '<td><%= flightcustomer._id %> </td>',
                            '<td><%= flightcustomer.json.customerNr %> </td>',
                            '<td><%= flightcustomer.json.name %></td>',
                            '<td><%= flightcustomer.json.street %></td>',
                            '<td><%= flightcustomer.json.zipCode %></td>',
                        '</tr>',
                    '<% }); %>',
                '</table>'
            ].join(''),

            //Populate the HTML template with content
            html = _.template(table, {flightcustomerlist : arr});

        //Put the generated HTML table into the DOM
        status.html(html);
    };
```

```
    //Create the optional options object passed to init
    var options = {};

    //JSONStore collections metadata
    var collections = {};

    //Define the 'people' collection and list the search fields
    collections[FLIGHTCUSTOMER_COLLECTION_NAME] = {

        searchFields : {customerNr: 'string', name: 'string', street: 'string', zipCode: 
'string'},

        //-- Start optional adapter metadata
        adapter : {
            name: 'FlightCustomer',
            add: 'addFlightCustomer',
            remove: 'deleteFlightCustomer',
            replace: 'updateFlightCustomer',
            load: {
                procedure: 'getFlightCustomers',
                params: [],
                key: 'flightcustomerlist'
            }
        }
        //-- End optional adapter metadata
    };

    //Initialize the people collection
    WL.JSONStore.init(collections, options)

    .then(function () {
        _logMessage(INIT_MSG);
    })
    .fail(function (errorObject) {
        _logMessage(errorObject.msg);
    });
    //find-all
    $('button#find-all').on('click', function () {

        //Get reference to the search field
        var limitField = $('input#find-limit'),
            offsetField =$('input#find-offset');

        //Get value from the search field
        var limit = parseInt(limitField.val(), 10) || '',
            offset = parseInt(offsetField.val(), 10) || '';

        //Create optional options object
        var options = {};

        //Check if limit was passed
        if (_.isNumber(limit)) {
            options.limit = limit;
        }

        //Check if offset was passed
        if (_.isNumber(offset)) {
            options.offset = offset;
        }
        try {
            //Alternative syntax:
            WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).findAll(options)
            .then(function (res) {
```

```
         _showTable(res);
      })
      .fail(function (errorObject) {
         _logMessage(errorObject.msg);
      });
      //Clear the input fields
      limitField.val('');
      offsetField.val('');

   } catch (e) {
      _logMessage(INIT_FIRST_MSG);
   }
});
//add
$('button#add-data').on('click', function () {

   //Get references to the input fields DOM elements
   var nameField = $('input#add-name'),
      streetField = $('input#add-street'),
      zipCodeField = $('input#add-zipCode');

   //Check if a name was passed
   if (nameField.val() == '') {
      return _logMessage(NAME_FIELD_EMPTY_MSG);
   }

   //Get values from the input fields
   var name = nameField.val() || '',
      street = streetField.val() || '',
      zipCode = zipCodeField.val() || '';

   //Prepare the data object
   var data = {};

   //Check if a name was passed
   if (name.length > 0) {
      data.name = name;
   }

   //Check if a street was passed
   if(street.length > 0) {
      data.street = street;
   }

   //Check if a zip Code was passed
   if(zipCode.length > 0) {
      data.zipCode = zipCode;
   }
   try {
      //Call add on the JSONStore collection
      WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).add(data)
      .then(function () {
         _logMessage(ADD_MSG);

      })
      .fail(function (errorObject) {
         _logMessage(errorObject.msg);
      });
      //Clear the input fields
      nameField.val('');
      streetField.val('');
      zipCodeField.val('');
```

```
    } catch (e) {
        _logMessage(INIT_FIRST_MSG);
    }
});
//replace
$('button#replace').on('click', function () {

    //Get references to the input fields DOM elements
    var nameField = $('input#replace-name'),
        streetField = $('input#replace-street'),
        zipCodeField = $('input#replace-zipCode'),
        idField = $('input#replace-id');

    //Get values from the input fields
    var name = nameField.val() || '',
        street = streetField.val() || '',
        zipCode = zipCodeField.val() || '',
        id = parseInt(idField.val(), 10) || '';

    //Check if an id was passed
    if (!_.isNumber(id)) {
        return _logMessage(ID_FIELD_EMPTY_MSG);
    }

    WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).findById(id)
    .then(function (res) {
        originaldocument = res[0];

        //Create the document object
        var doc = {
            _id : id,
            json : {}
        };

        // populate Customer number which is read-only in the mobile app
        if (originaldocument != null) {
            doc.json.customerNr = originaldocument.json.customerNr;
        }

        //Check if a name was passed
        if (name.length > 0) {
            doc.json.name = name;
        }else
        {
            doc.json.name = originaldocument.json.name;
        };

        //Check if a street was passed
        if (street.length > 0) {
            doc.json.street = street;
        }else
        {
            doc.json.street = originaldocument.json.street;
        };

        //Check if a zipCode was passed
        if (zipCode.length > 0) {
            doc.json.zipCode = zipCode;
        }else
        {
            doc.json.zipCode = originaldocument.json.zipCode;
        };
        try {

            WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).replace(doc)

            .then(function () {
               _logMessage(REPLACE_MSG);
            })

            .fail(function (errorObject) {
               _logMessage(errorObject.msg);
            });

            //Clear the input fields
            nameField.val('');
            streetField.val('');
            zipCodeField.val('');
            idField.val('');

        } catch (e) {
            _logMessage(INIT_FIRST_MSG);
        }
    })
    .fail(function (errorObject) {
        return _logMessage(errorObject.msg);
    });
 });
 //remove
 $('button#remove-id-btn').on('click', function () {

    //Get reference to the id field
    var idField = $('input#remove-id');

    //Get value from the search field
    var id = parseInt(idField.val(), 10) || '';

    //Check if an id was passed
    if (!_.isNumber(id)) {
        return _logMessage(ID_FIELD_EMPTY_MSG);
    }

    //Build the query object
    var query = {_id: id};

    //Build the options object, if exact: true
    //is not passed fuzzy searching is enabled
    //that means id: 1 will match 1, 10, 100, ...
    var options = {exact: true};

    try {

        WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).remove(query, options)

        .then(function (res) {
            _logMessage(REMOVE_MSG + res);
        })

        .fail(function (errorObject) {
            _logMessage(errorObject.msg);
        });

        //Clear the input fields
        idField.val('');
```

```
    } catch (e) {
        _logMessage(INIT_FIRST_MSG);
    }
});
//load
$('button#load').on('click', function () {

    try {

        WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).clear()

        .then(function (res) {
            _logMessage(REMOVE_COLLECTION_MSG + res);

            WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).load()

            .then(function (res) {
                _logMessage(LOAD_MSG + res);
            })

            .fail(function (errorObject) {
                _logMessage(errorObject.msg);
            });
        })

        .fail(function (errorObject) {
            _logMessage(errorObject.msg);
        });
    } catch (e) {
        _logMessage(INIT_FIRST_MSG);
    }

});
//getPushRequired
$('button#get-push-required').on('click', function () {

    try {

        WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).getPushRequired()

        .then(function (res) {
            _showTable(res);
        })

        .fail(function (errorObject) {
            _logMessage(errorObject.msg);
        });

    } catch (e) {
        _logMessage(INIT_FIRST_MSG);
    }

});
//push
$('button#push').on('click', function () {

    try {

        WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).push()

        .then(function (res) {

            if (_.isArray(res) && res.length < 1) {
                //Got no errors pushing the adapter to the server
                _logMessage(PUSH_MSG);

             } else {
                //The array contains error responses from the adapter
                _logMessage(PUSH_MSG_FAILED + _.first(res).res.errorCode);
             }

          })

          .fail(function (errorObject) {
             _logMessage(errorObject.msg);
          });

       } catch (e) {
          _logMessage(INIT_FIRST_MSG);
       }

    });
    //count
    $('button#count').on('click', function () {

       try {

          WL.JSONStore.get(FLIGHTCUSTOMER_COLLECTION_NAME).count()

          .then(function (res) {
             _logMessage(COUNT_MSG + res);
          })

          .fail(function (errorObject) {
             _logMessage(errorObject.msg);
          });

       } catch (e) {
          _logMessage(INIT_FIRST_MSG);
       }
    });
} (WL, WLJQ, WL_) );
}//end wlCommonInit
```

Now, the hybrid mobile application is ready to be built and deployed. These tasks are part of the IBM MobileFirst Studio standard features and are described in 1.3, “Testing the mobile application” on page 21.

3.3 Testing the mobile application

The mobile application can be tested by using the MobileFirst Console and a web browser. The Chrome browser is used in this scenario to access `Preview as Common Resources` link in the MobileFirst Console. The test cases for this application are as follows:

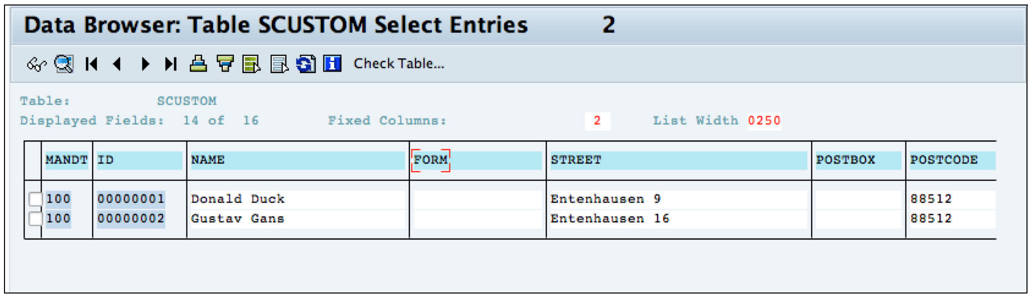
- Initial data load from the SAP ERP system.
- Create a new flight customer record and push the change back to the SAP system.
- Change an existing flight customer record and push the update back to the SAP system.

The following sections describe the test cases. You can use the sample code that is delivered with this Redpaper publication and test it directly on your local development workstation where IBM MobileFirst Studio is installed.

3.3.1 Initial data load from the SAP ERP system

To run this test case, complete the following steps:

1. Determine which flight customer records are available on the target SAP ERP system. You can review the records by using the SAP transaction `SE16` and viewing the `SCUSTOM` table, which displays a result set (Figure 3-13).



Data Browser: Table SCUSTOM Select Entries 2

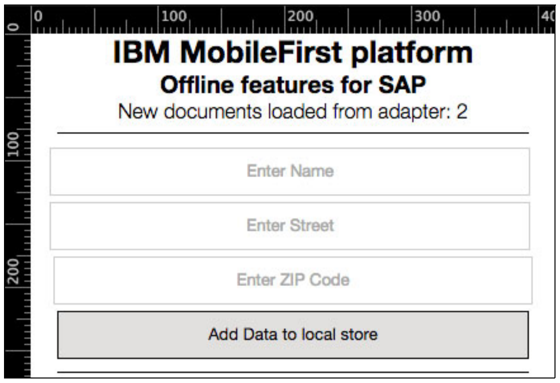
```
Table:          SCUSTOM
Displayed Fields:  14 of  16        Fixed Columns:            2     List Width 0250
```

MANDT	ID	NAME	FORM	STREET	POSTBOX	POSTCODE
100	00000001	Donald Duck		Entenhausen 9		88512
100	00000002	Gustav Gans		Entenhausen 16		88512

Figure 3-13 Reviewing existing flight customer records in the SAP ERP system

2. Start the mobile application by using the IBM MobileFirst Console, and click **Pull business data from SAP ERP**.

3. Review the status line, which displays the number of flight customer records that were loaded (Figure 3-14).



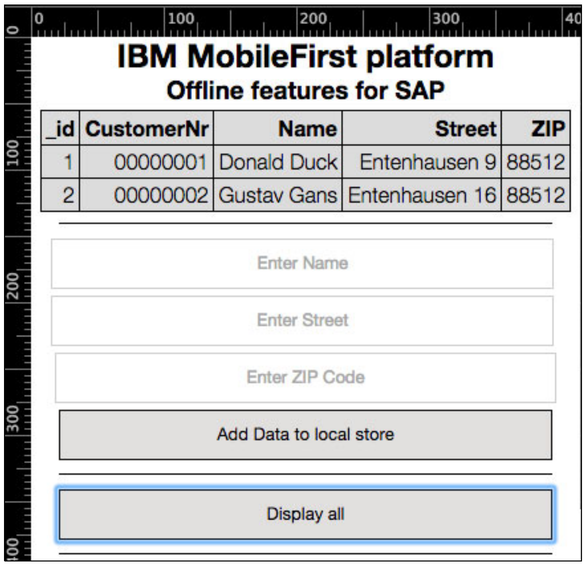
IBM MobileFirst platform

Offline features for SAP

New documents loaded from adapter: 2

Figure 3-14 Successfully loaded two flight customer records

4. Click **Display all** to visualize a list of all flight customer records that are now available locally in the mobile device (Figure 3-15). The flight customer records are displayed in the status area.


IBM MobileFirst platform

Offline features for SAP

_id	CustomerNr	Name	Street	ZIP
1	00000001	Donald Duck	Entenhausen 9	88512
2	00000002	Gustav Gans	Entenhausen 16	88512

Figure 3-15 Flight customer records available offline on local device

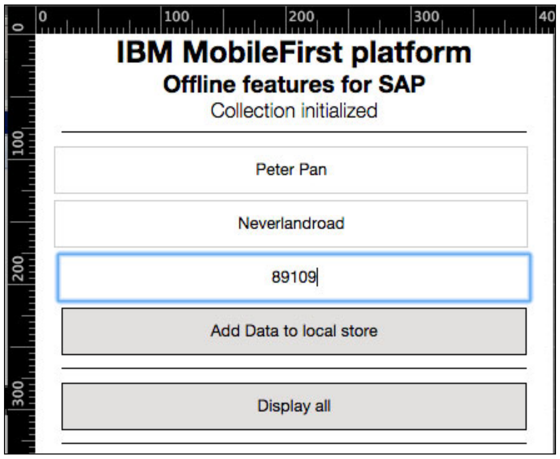
In this example, the local JSONStore component stores two flight customer records. Each time the load action is triggered, it replicates all available records.

You may narrow the returned result set by defining granular filter options. This functionality, however, is not implemented in the example included in this paper.

3.3.2 Create a new flight customer record and push the change back to the SAP system

To run this test case, complete the following steps:

1. Start the mobile application from the **MobileFirst Console**.
2. Enter values in the input fields for name, street, and postal code, as shown in Figure 3-16.


IBM MobileFirst platform

Offline features for SAP

Collection initialized

Peter Pan

Neverlandroad

89109

Figure 3-16 Entering flight customer data

3. Click **Add Data to local store**.
4. Check the status message to ensure that the flight customer record is added locally. The status line displays the results (Figure 3-17).

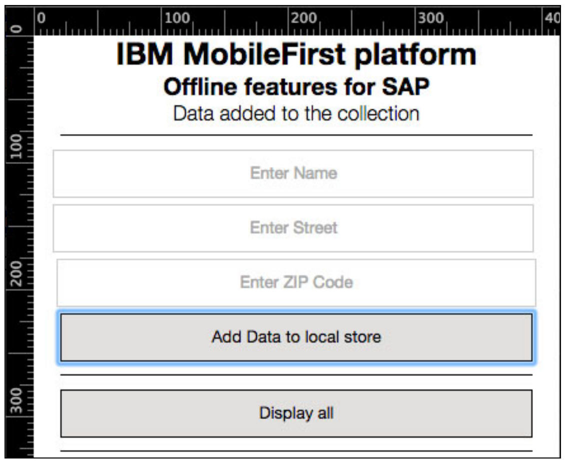
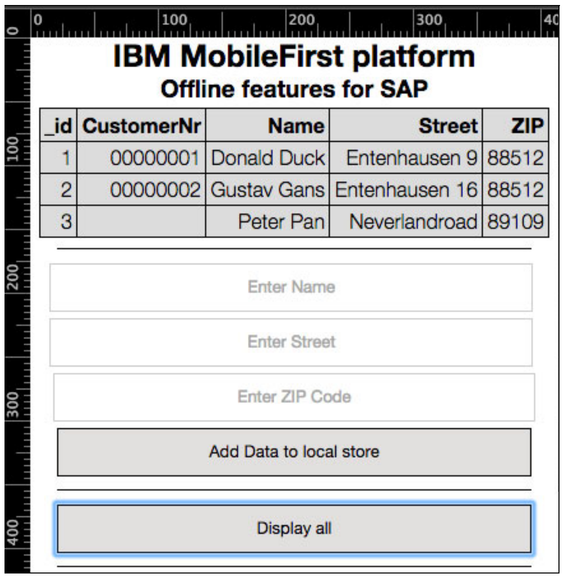


Figure 3-17 Flight customer record added successfully to local storage

5. Click **Display all** to display the content of the local flight customer records (Figure 3-18).

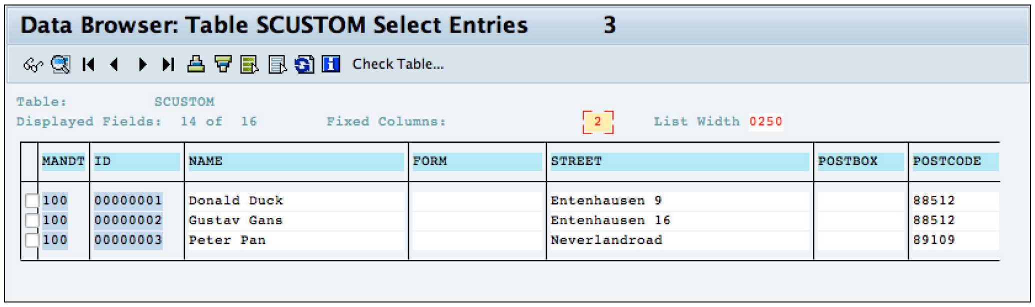


_id	CustomerNr	Name	Street	ZIP
1	00000001	Donald Duck	Entenhausen 9	88512
2	00000002	Gustav Gans	Entenhausen 16	88512
3		Peter Pan	Neverlandroad	89109

Figure 3-18 Displaying all local flight customer records

6. The new record has no customer number because this number is generated by the SAP system. Click **Push business data to SAP ERP** to start synchronization.

7. Check that the new flight customer record arrived in the SAP ERP system and that the corresponding customer number is assigned (Figure 3-19).



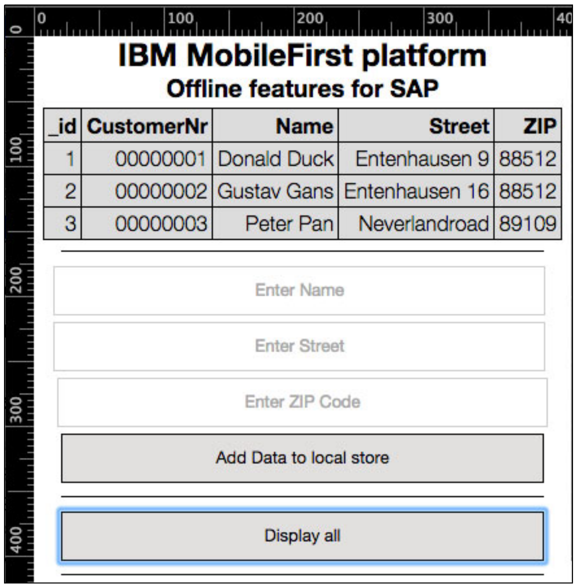
```
Table:            SCUSTOM
Displayed Fields:  14 of  16        Fixed Columns:            2      List Width 0250
```

MANDT	ID	NAME	FORM	STREET	POSTBOX	POSTCODE
100	00000001	Donald Duck		Entenhausen 9		88512
100	00000002	Gustav Gans		Entenhausen 16		88512
100	00000003	Peter Pan		Neverlandroad		89109

Figure 3-19 New flight customer record in the SAP system

8. After a successful response in the status line, triggering a full load of the data again is possible by clicking **Pull business data from SAP ERP**.

9. To display the results, click **Display all**. The content of the local JSONStore is displayed in the status line (Figure 3-20).



_id	CustomerNr	Name	Street	ZIP
1	00000001	Donald Duck	Entenhausen 9	88512
2	00000002	Gustav Gans	Entenhausen 16	88512
3	00000003	Peter Pan	Neverlandroad	89109

Figure 3-20 Displaying updated flight customer records

The test case is complete. It demonstrated that generating a new flight customer record on the mobile device and synchronizing the data with the SAP system are possible. In this scenario, the SAP system is responsible for generating a unique customer number.

3.3.3 Change an existing flight customer record and push the update back to the SAP system

This test case involves changing an existing flight customer record by changing the street name and pushing the update back to the SAP ERP system.

To run this test case, complete the following steps:

1. Start the mobile application with the **MobileFirst Console**.
2. Enter the JSONStore unique ID of the flight customer record you want to change and the new street name, as shown in Figure 3-21.

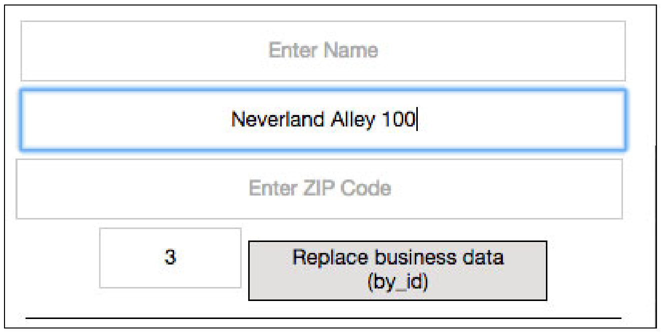
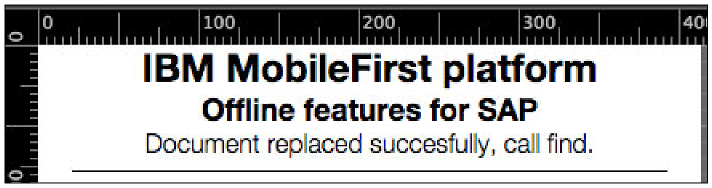


Figure 3-21 Entering a new street name and JSON store unique id

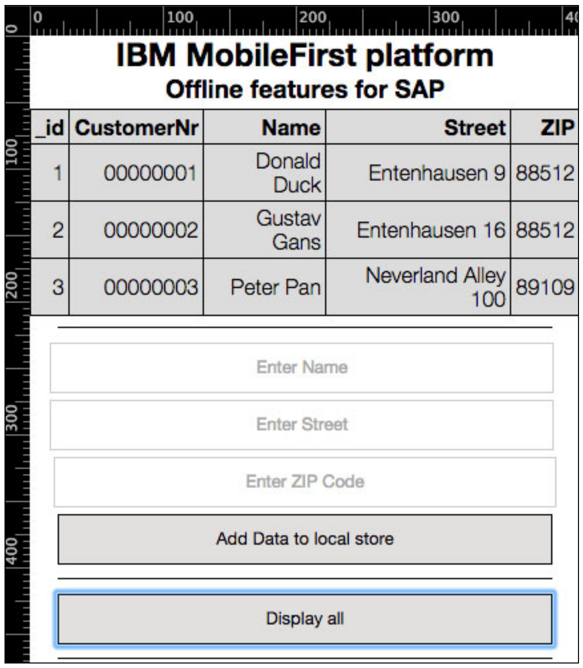
3. Click **Replace business data (by_id)**.
4. Check the status line for a positive response (Figure 3-22).



IBM MobileFirst platform
Offline features for SAP
Document replaced succesfully, call find.

Figure 3-22 Status line displaying the result

5. Click **Display all** to verify that the change is locally persisted (Figure 3-23).



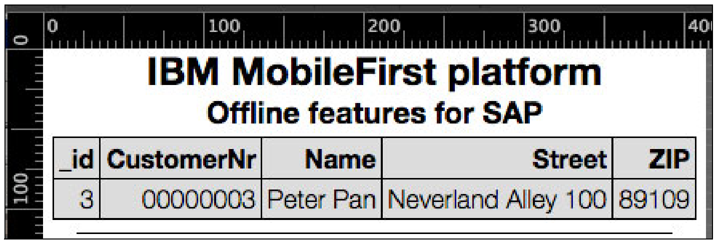
IBM MobileFirst platform

Offline features for SAP

_id	CustomerNr	Name	Street	ZIP
1	00000001	Donald Duck	Entenhausen 9	88512
2	00000002	Gustav Gans	Entenhausen 16	88512
3	00000003	Peter Pan	Neverland Alley 100	89109

Figure 3-23 Verify changed flight customer record

6. Click **Get documents for sync** to see the documents that are marked for transfer to the SAP system. The result set looks like the one shown in Figure 3-24.



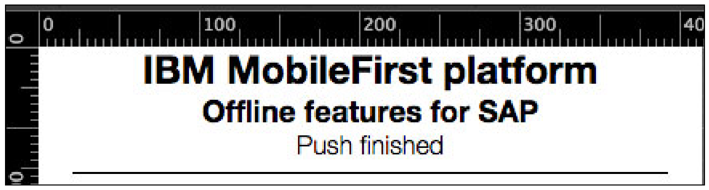
IBM MobileFirst platform

Offline features for SAP

_id	CustomerNr	Name	Street	ZIP
3	00000003	Peter Pan	Neverland Alley 100	89109

Figure 3-24 Flight customer records with local changes

7. Click **Push business data to SAP ERP** to trigger synchronization with the SAP system. The status line is updated (Figure 3-25).



IBM MobileFirst platform

Offline features for SAP

Push finished

Figure 3-25 Pushing changes to the SAP ERP completed

8. Verify that the changed flight customer record arrived in the SAP system (Figure 3-26).

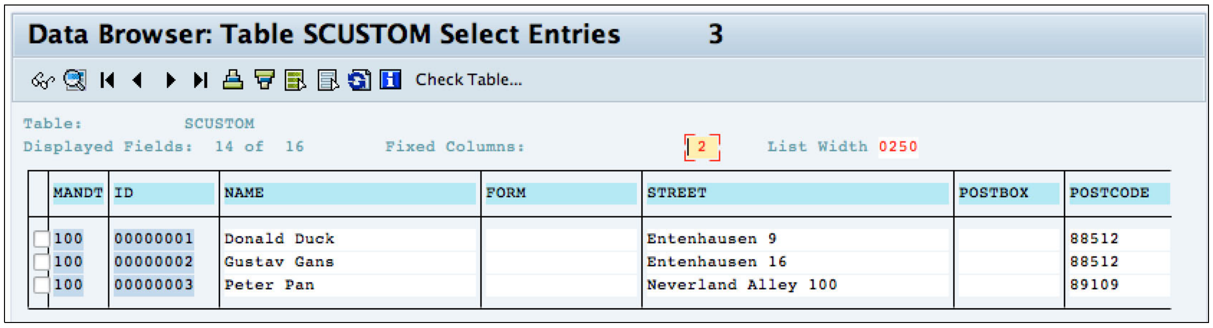


Table: SCUSTOM
Displayed Fields: 14 of 16 Fixed Columns: 2 List Width 0250

MANDT	ID	NAME	FORM	STREET	POSTBOX	POSTCODE
100	00000001	Donald Duck		Entenhausen 9		88512
100	00000002	Gustav Gans		Entenhausen 16		88512
100	00000003	Peter Pan		Neverland Alley 100		89109

Figure 3-26 Updated flight customer record in SAP

This test case demonstrated that changing flight customer records locally in the mobile device and synchronizing the changes with the SAP ERP system are possible.

3.4 Additional features and capabilities

There are more enhanced features and capabilities in the IBM MobileFirst Platform JSONStore component that are not described in this paper. Of special interest for enterprise customers are the enhanced transaction capabilities to run synchronizations with transactional safety and trigger roll backs when errors occur.

Another powerful enterprise feature is the encryption mechanism, which is not described in this paper in detail. For details about the features and functions of the IBM MobileFirst Platform JSONStore capabilities, see IBM MobileFirst Platform Foundation V6.3.0 documentation at IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSHS8R_6.3.0/wl_welcome.html

A

Samples included with this paper

This appendix describes the summary information for the samples provided with this Redpaper publication.

Also included are instructions to import the sample projects for each chapter into the IBM MobileFirst Studio workspace.

Disclaimer: IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE OR UP-TO-DATE. IBM DOES NOT WARRANT, REPRESENT OR IMPLY RELIABILITY, SERVICEABILITY OR FUNCTION OF THE CODE. IBM IS UNDER NO OBLIGATION TO UPDATE CONTENT NOR PROVIDE FURTHER SUPPORT.

ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER. IBM EXPRESSLY DISCLAIMS TO THE FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR REPRESENTATIONS, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL PROPERTY RIGHTS.

YOU UNDERSTAND AND AGREE THAT YOU USE THESE MATERIALS, INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.

IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY TYPE WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

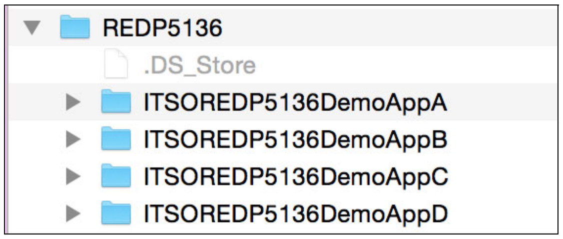
THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER LEGAL THEORIES.

Importing the samples from the ZIP file

Complete the following steps to work with the sample projects included with this book:

1. Download and extract the `REDP5136.zip` file as described in Appendix B, “Additional material” on page 91.

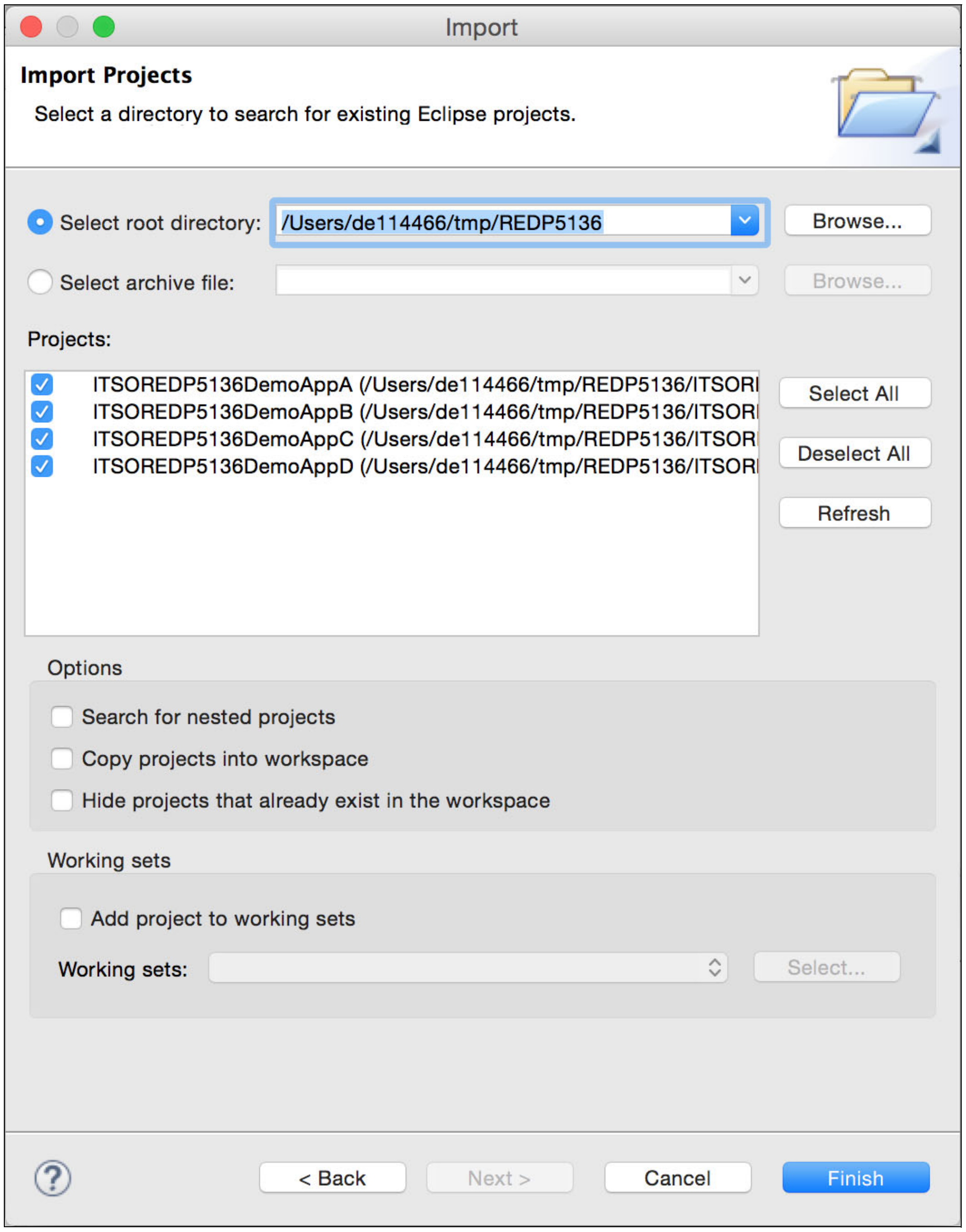
   You will have the folder structure shown in Figure A-1.



REDP5136
.DS_Store
ITSOREDP5136DemoAppA
ITSOREDP5136DemoAppB
ITSOREDP5136DemoAppC
ITSOREDP5136DemoAppD

Figure A-1 Folder structure of IBM MobileFirst Studio export archive

2. Import all project directories into the IBM MobileFirst Studio workspace by using the Eclipse import wizard (Figure A-2 on page 89).



Import

Import Projects

Select a directory to search for existing Eclipse projects.

[✓ Select root directory:] /Users/de114466/tmp/REDP5136

Select archive file:

Projects:

- ✓ ITSOREDP5136DemoAppA (/Users/de114466/tmp/REDP5136/ITSORI
- ✓ ITSOREDP5136DemoAppB (/Users/de114466/tmp/REDP5136/ITSORI
- ✓ ITSOREDP5136DemoAppC (/Users/de114466/tmp/REDP5136/ITSORI
- ✓ ITSOREDP5136DemoAppD (/Users/de114466/tmp/REDP5136/ITSORI

Options

- ☐ Search for nested projects
- ☐ Copy projects into workspace
- ☐ Hide projects that already exist in the workspace

Working sets

- ☐ Add project to working sets

Working sets:

Figure A-2 Import sample projects into workspace

3. Click **Finish** to start the import process.

Where the samples are used

This paper describes four mobile applications that demonstrate the capabilities in practical examples. All chapters in this paper are self-contained so there is no need to import a sample to start the development work. The samples are included for reference only to ensure you have a running example.

Table A-1 lists the sample folders and the corresponding location (chapter or section) in this paper where the sample is described.

Table A-1 Sample projects included with this paper

Sample folder name	Location where the sample is used	Description
`ITSOREDP5136DemoAppA`	Chapter 1, "Mobile authentication with SAP Business Suite" on page 1.	Implement a custom login module to authenticate with an SAP ERP system.
`ITSOREDP5136DemoAppB`	2.2, "Using the IBM MobileFirst Cast Iron adapter for SSO authentication with SAP Business Suite" on page 33.	Implement an SSO scenario using IBM Cast Iron as the integration component.
`ITSOREDP5136DemoAppC`	2.3, "Using the IBM MobileFirst SAP NetWeaver Gateway adapter for SSO authentication with SAP Business Suite" on page 45.	Implement an SSO scenario using SAP NetWeaver Gateway as the integration component.
`ITSOREDP5136DemoAppD`	Chapter 3, "Offline SAP business data storage and synchronization in mobile scenarios" on page 55.	Implement an offline scenario that stores business data on the local mobile device using the offline store capabilities of the IBM MobileFirst Platform.

B

Additional material

This paper refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your web browser at:

`ftp://www.redbooks.ibm.com/redbooks/REDP5136`

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redpaper form number, REDP5136.

Using the web material

The additional Web material that accompanies this paper includes the following file:

File name	*Description*
REDP5136.zip	Compressed code samples

System requirements for downloading the Web material

The Web material requires the following system configuration:

Hard disk space:	2 MB minimum
Operating System:	Windows or Linux

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the `.zip` file into this folder.

See Appendix A, “Samples included with this paper” on page 87 for a description of where the samples are used in this paper.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Some publications referenced in this list might be available in softcopy only.

- *Extending IBM Business Process Manager to the Mobile Enterprise with IBM Worklight*, SG24-8240
- *IBM MobileFirst Strategy Software Approach*, SG24-8191
- *IBM Software for SAP Solutions*, SG24-8230
- *Securing Your Mobile Business with IBM Worklight*, SG24-8179

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- IBM MobileFirst Platform Foundation V6.3.0 documentation

  http://www.ibm.com/support/knowledgecenter/SSHS8R_6.3.0/wl_welcome.html
- IBM MobileFirst Platform announcement letter

  http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=AN&subtype=CA&htmlfid=897/ENUS214-344&appname=USN

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Extending SAP Solutions to the Mobile Enterprise with IBM MobileFirst Platform Foundation

Redpaper™

Learn how to enable mobile access to SAP business systems

Implement SSO for SAP systems with ready-to-use adapters

Exploit offline capabilities and sychronize with enterprise SAP data

SAP solutions provide a rich set of composable application modules, and configurable functional capabilities that are expected from a comprehensive enterprise business application software suite.

Enabling mobile access to SAP business functions and data for enterprise clients, employees, and business partners is a typical requirement for SAP projects.

This IBM Redpaper publication describes the benefits of the IBM MobileFirst Platform Foundation security framework, which is an essential building block of the IBM MobileFirst platform. This paper also discusses key capabilities in IBM MobileFirst Platform to authenticate with SAP business systems from mobile applications.

The scenarios in this paper demonstrate these features:

- How to develop a custom login module with IBM MobileFirst to authenticate mobile user access to an SAP enterprise resource planning (ERP) system.
- How to implement single sign-on (SSO) authentication in a mobile application to access SAP business systems using a pre-built adapter for IBM Cast Iron included with IBM MobileFirst and the ready-to-use adapter for SAP NetWeaver Gateway, also included with IBM MobileFirst.
- How to take advantage of the offline capabilities included in IBM MobileFirst Platform Foundation V6.3 to store business data locally, act on this data, and synchronize the data with the originating SAP ERP system.

This paper is for mobile application developers and technical consultants who design and build systems of engagement to interact with SAP solutions in the heterogeneous enterprise.

REDP-5136-00