webMethods Application Platform Tutorial

Version 9.9

October 2015

WEBMETHODS

This document applies to webMethods Application Platform Version 9.9 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: AP-TUT-99-20151015

CONTENTS

1 Introduction

This tutorial provides step-by-step walk-through examples of how to create different types of Application Platform projects in Designer and deploy them to an Integration server runtime (IS) and also My webMethods Server runtime (MWS). It also shows how to use the Application Platform Bundle Manager tool to create OSGI bundles from regular jars and how to use the Shared Bundles tool to add third-party bundles to the build classpath.

1.1 Goals of This Tutorial

There are two parts to this tutorial. Part 1 shows how to use Application Platform with IS and part 2 shows how to use Application Platform with MWS

Part 1 will teach you how to:

- Create an Application Platform project with Java code that calls IS services
- Create an Application Platform project with IS services that call your Java code
- Start and stop the Application Platform Server from Designer
- Generate Java code from Integration Server services
- Use Application Platform Bundle Manager to create wrapper bundles
- Use Application Platform Shared Bundles tool to add third party bundles to the build classpath
- Publish a project to Integration Server
- Unpublish a project from Integration Server
- Invoke services in the published projects

Part 2 shows how to:

- How to make use of App Platform project services from an MWS CAF web project

1.2 Requirements for This Tutorial

To complete this tutorial it is assumed that (refer to the webMethods Application Platform User's Guide on how to set these up):

- At least the typical Application Platform Development installation has been installed in your local host machine. This means that Designer, Integration Server, and Application Platform are installed. You can choose to install Integration Server and Designer in different installation root directories if you choose not to use the typical installation.
    - See how to install Designer at http://documentation.softwareag.com/
- You are familiar with basic Eclipse workbench mechanisms, such as views and perspectives.
- The Application Platform runtime environment is configured properly
    - See the webMethods Application Platform User's Guide

1.3 Sample Code and Third-Party Jars and Zip Files

- Required code samples, third-party jars and zip files are uploaded as attachments at the following address at the TECHcommunity website:
  http://techcommunity.softwareag.com/pwiki/-/wiki/Main/Application+Platform

2 Understanding Application Platform

webMethods Application Platform is a tool for Java developers that use Software AG Designer, webMethods Integration Server, My webMethods Server and other Software AG tools. It is used to easily create Java business logic, deploy the logic into the SAG common runtime and invoke the logic from a variety of front ends, like the web. The target users should be familiar with Eclipse, Spring, Hibernate, and Tomcat.

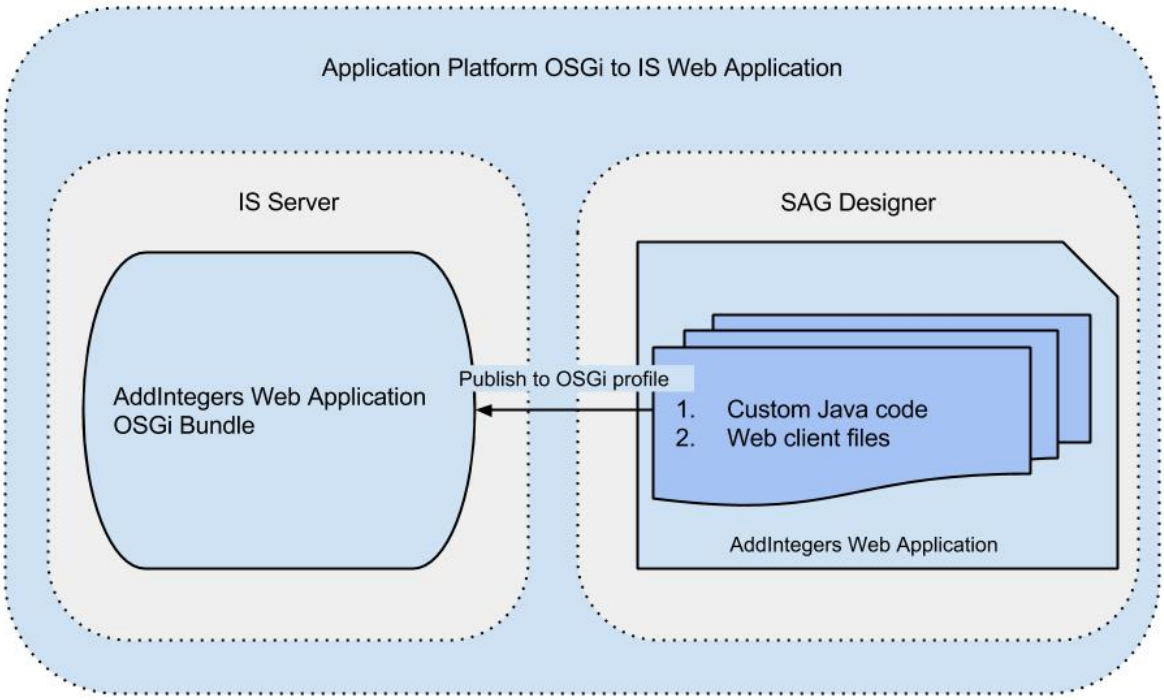
3 Creating Your First Java Project - a Web Application

This chapter gives step-by-step instructions on how to create an Application Platform Java web project example.

3.1 Architecture of the Example Application

The example application consists of three main components: the AddIntegers web client, created by using JavaServer pages technology; an AddIntegers java class that does the math calculation of adding two integers together and returning the result; and the AddIntegers OSGi Java application bundle, which is deployed to the Integration Server container's OSGi profile by the user.

Figure 3-1: Architecture of the AddIntegers web application



The AddIntegers web client is a JavaServer pages application that displays the current date on the application's webpage at the top, displays two text fields for the user to enter two integers, and then, when the user clicks the plus button, it adds the two integers together and displays the result on a label on the web page.

The AddIntegers OSGi bundle contains the web client's JSP and HTML web application files and the custom Java code, created by the user.

3.1.1 Tiers in the Example Application

The AddIntegers web client is the web tier component for this web application; the AddIntegers OSGi bundle is the business tier; and the user's web browser is the client tier component.

3.2 Coding the Web Project Example

This section describes how to create the AddIntegers example web project.

3.3 Getting Started

Before you can start coding the example, you need to perform the following steps:

1. Install the following products in the same root folder on your computer (for installation instructions, see Installing webMethods and Intelligent Business Operations Products):
    a. Install Software AG Designer.
    b. Install webMethods Integration Server.
    c. Install webMethods Application Platform on your computer.
2. Start Designer

3.4 Creating the Project

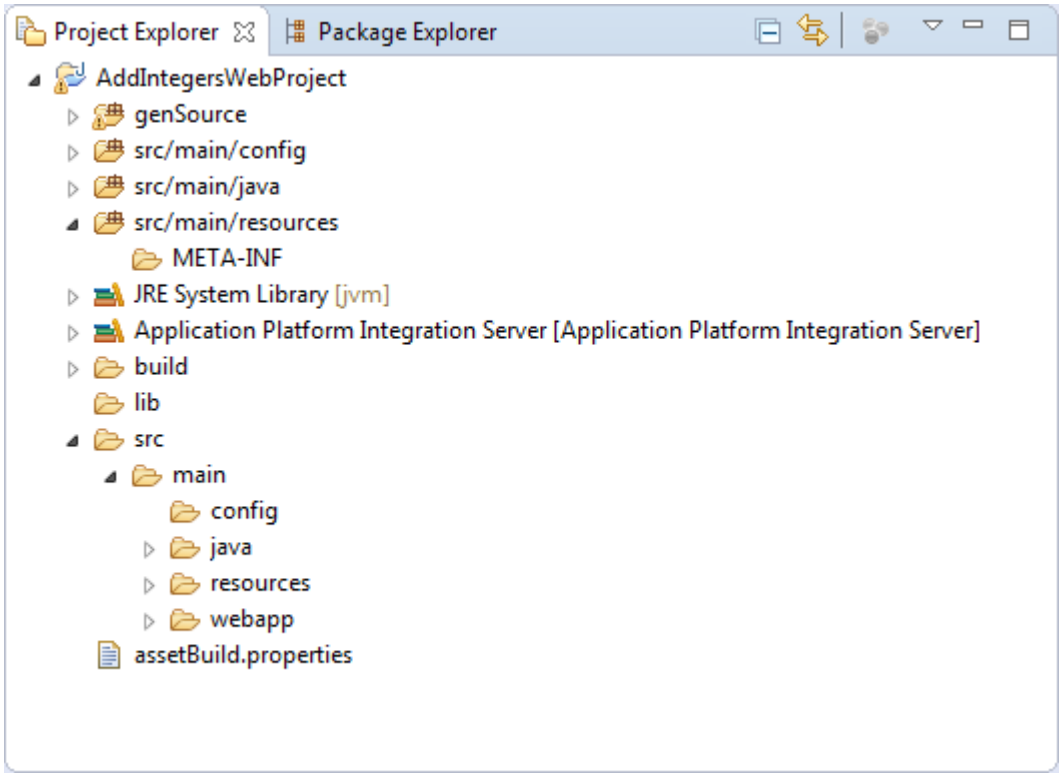
1. In Designer close the Software AG welcome page if it is open - the Welcome page is displayed the first time you open a brand new Designer or if you create a new workspace.
2. In Designer's Window menu, choose Open Perspective.
3. Select Application Platform in the submenu that opens.
    a. If Application Platform is not available in the submenu, choose Other, and then select Application Platform
    b. An Application Platform perspective information dialog opens, allowing you to configure an Application Platform runtime environment.
    c. Clicking the Yes button will open the Server Runtime Environments page in the Preferences dialog.
    d. Click the Add... button to open the New Server Runtime Environment dialog and proceed as in 9.b below.
    e. Click OK in the Preferences dialog to close it when finished and open the Application Platform perspective.
4. From the File menu, choose New and then Web Project to open the Application Platform Core Web UI Template dialog; Alternatively:

a. From the File menu, choose New and then Other....
b. Select Software AG > App Platform > Web Project in the “Select a wizard” dialog that opens and click Next to create a new web project.

5. Enter a project name and click Next to go to the Project Facets page. Enter AddIntegersWebProject as the name for our sample project.
6. Confirm that Application Platform Web UI Preset is the selected configuration
7. Confirm that the following project facets are preselected for you:
   a. Java
   b. Application Platform Core and Application Platform Web both under SoftwareAG Application Platform.
8. Check Integration Server Extensions facet - it is unchecked by default; you will need this for the second project, which is an extension of this project.
9. Click on the Runtimes tab on the right hand of the dialog and select the “Application Platform Integration Server” checkbox if it is not selected
   a. If the "Application Platform Integration Server" checkbox is not available then click on the New... button to open the New Server Runtime Environment dialog.
   b. Expand the Software AG branch in the window and select "Application Platform Integration Server."
   c. Select the "Create a new local server" checkbox.
   d. Click the Next > button to go to the "New Application Platform Integration Server Runtime" page.
   e. Select the required JRE.
   f. Browse to select your Integration Server Installation root directory.
   g. Click Next > to go to the "New Application Platform Integration Server Server" page.
   h. Make any required changes to the displayed property values, if necessary. Otherwise, use the default values.
   i. Click the Finish button to complete creating the new server runtime environment.
10. Click Next to go to the Java page.
11. We will use the default src folder for this example project but if you want to use a different source folder perform the following:
    a. Delete the existing src folder.
    b. Click on the Add Folder... button to add a new source folder.
    c. Enter a new name.
    d. Click Ok to add the folder to the list of source folders on the build path.
12. Click Next to go to the Application Platform IS Facet page
13. Make sure “Include Generated Source path” checkbox is selected.
14. Leave Generated Source Path as is and click Next > to go to the Application Platform Web Facet page.

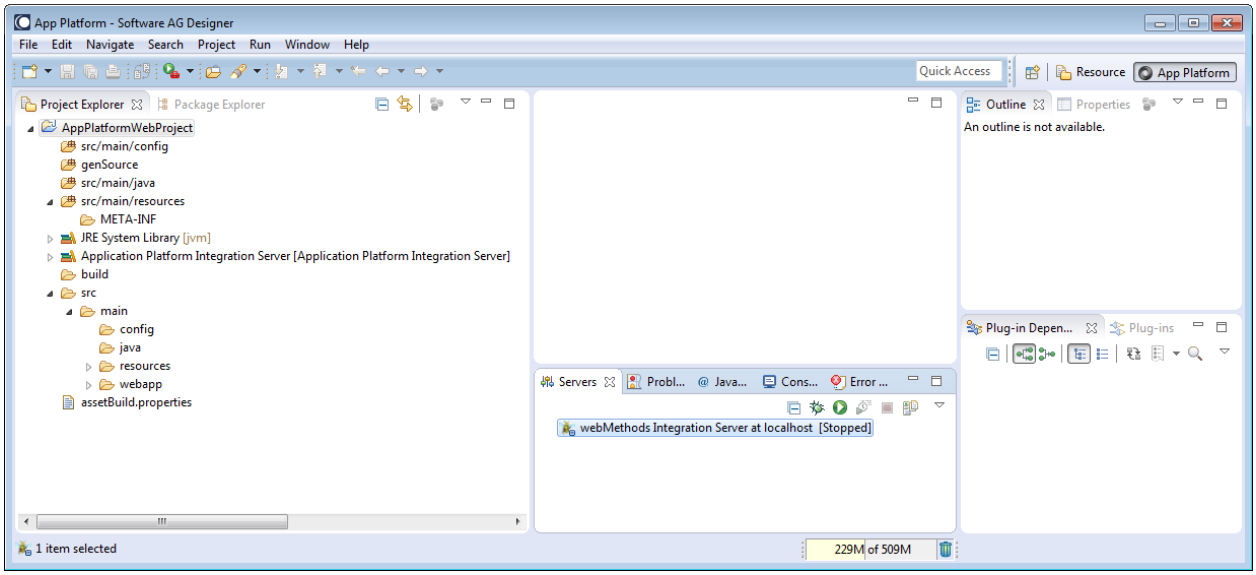
15. For this example leave the Web Context Directory as the default but know that you can change it if you wish.
16. Click the Finish button to create the web project. You should now see the project created in the Projects Explorer view with a structure like the one in figure 3-2 below.

Figure 3-2: Sample view of a new AddIntegersWebProject in Project Explorer



17. You should also see “webMethods Integration Server at localhost [Stopped]” added to the Servers view as shown in figure 3-3 below.

Figure 3-3: Sample view of the AppPlatform perspective just after creating the AddIntegersWebProject showing the project in the Project Explorer view and “webMethods Integration Server at localhost [Stopped]” added to the Servers view



3.5 Creating your Java code

Now add your Java code under the src/main/java folder:

1. Expand src/main/java.
2. Right-click src/main/java, select New > Package to open the New Java Package dialog.
3. Type com.addints in the Name field.
4. Click Finish to create the package.
5. Right-click the com.addints package and select New > Class to open the New Java Class dialog.
6. Type AddIntegers in the Name field.
7. Leave the rest of the values as default and click the Finish button to create the AddIntegers.java class, which should also open in the Java editor in the Eclipse Workbench.
8. Type the following code in the editor:

```
// Copyright (c) 2014-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.
    // Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.
    package com.addints;

    import java.text.*;
    import java.util.Calendar;
    import java.util.Date;
```

```
import com.sun.org.apache.xerces.internal.impl.xpath.regex.ParseException;

public class AddIntegers {

   public Date getCurrentDate(){

          DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

          //get current date time with Calendar()
             Calendar cal = Calendar.getInstance();
             System.out.println(dateFormat.format(cal.getTime()));
        // Get results from outputObject

          Date date = new Date();
          try{

                date = dateFormat.parse(dateFormat.format(cal.getTime()));
          }catch(ParseException | java.text.ParseException pe){
                pe.printStackTrace();
          }

          return date;
   }

   public String addTwoIntegers(String num1, String num2){

          Integer result = new Integer(num1) + new Integer(num2);

        return result.toString();
   }
}
```

3.6 Creating the Example Web Client

Now add the web client code:

1. Navigate to the src --> main --> webapp folder and right click the webapp folder.
2. Select New > Other… to open the Select a wizard dialog.
3. Expand the Web branch and select JSP File to open the New JSP File dialog.
4. Enter addIntegers.jsp in the File Name text field.
5. Select Next > to go to the “Select JSP Template” page.

6. Unselect the “Use JSP template” checkbox and click the Finish button.
7. The addIntegers.jsp file should now be displayed under src/main/webapp and should also be open in the main window.
8. Type the following code in the addIntegers.jsp file:

```
<!DOCTYPE html>
<html>
	<head>
		<link type="text/css" rel="stylesheet" href="addintegers.css">
		<title>Add Two Numbers</title>
	</head>
	<body>
		<%@page	import="com.addints.AddIntegers"%>
		<h1>Add Two Numbers</h1>

		<%AddIntegers addInts = new AddIntegers();%>

		<%! String num1="", num2="", result = ""; %>

		<%
			if(request.getParameter("add") != null) {
				num1=request.getParameter("integer1");
				num2=request.getParameter("integer2");
			result=addInts.addTwoIntegers(request.getParameter("integer1"),
request.getParameter("integer2"));
			}else if(request.getParameter("clear") != null){
				num1="";
				num2="";
				result="";
			}
		%>

		<div class="_addintegers">
			<form id="addItemForm" action="addIntegers.jsp" method="POST">

				<label for="currentdate"><%=addInts.getCurrentDate()%></label>
				<br/>
				<input class="txtfield" type="text" name="integer1"
value=<%=num1%>>
				<label for="add">+</label>
				<input class="txtfield" type="text" name="integer2"
value=<%=num2%>>
```

```
                        <label for="equal">=</label>
                        <input class="txtfield" type="text" name="result"
value="<%=result%>" disabled>
                        <br/>
                        <input type="submit" name="add" value="Add">
                        <input type="submit" name="clear" value="Clear">
                    </form>
            </div>

    </body>
</html>
```

Note: If you copied and pasted this code, you may have to correct any words that may have been split by dashes - e.g., re-sult, val-ue or re-quest. These dashes were automatically added by Microsoft Word when the JSP code was copied to this file. The Eclipse JSP editor may display a warning message as follows, "Undefined attribute name (val-ue)".

9. Right-click the webapp folder again and select New > Other… to open the Select a wizard dialog.
10. Expand the Web branch and select CSS File.
11. Click Next > to go to the CSS page.
12. Type addintegers for the file name.*Note that it is case sensitive.*
13. Click Next > to go to the "Select CSS Template" page.
14. Unselect the "Use CSS Template" checkbox and click the Finish button.
15. The addintegers.css file should now appear in the src/main/webapp folder.
16. Type the following code in the addintegers.css file:

```
h1 {
    margin: 4em 0em 1em 0em;
    background-color: #006699;
    color: #FFFFFF;
    text-align: center;
}

h3 {
    margin: auto;
    text-align: center;
}
```

```
._addintegers {
    margin: auto;
    background-color: #F8F8F8;
    text-align: center;
}

#results_label {
    background-color: #F8F8F8;
    text-align: center;
}

.additem_div {
    margin: auto;
    text-align: center;
    border-radius: 5em;
}

form {
    display: inline-block;
}

.home_page {
    display: inline-block;
    margin:auto;
    text-align:center;
}

label {
    display: inline-block;
    width: 9em;
    text-align:center;
}
.txtfield{
    display: inline-block;
    width: 11em;
}
```

3.7 Compiling the Project

Designer can be set to build projects automatically. You can also publish projects manually. If you are compiling manually:

1. In Designers Project menu select the “Build Automatically” menu item’.
2. Choose Clean… in the same Project menu to open the Clean dialog.
3. Choose the “Clean projects selected below” radio button and select only the AddIntengersWebProject checkbox to only build this project.
4. Click OK to build the project.

3.8 Publishing the Project to Integration Server

Publish the AddIntegersWebProject to the Integration Server.

1. Go to the Servers view.
2. If the server is not already started, start it by right-clicking on “webMethods Integration Server at localhost [Stopped]” and clicking Start on the popup menu.
    a. The focus may switch to the Console view until the server has started, and then switch back to the Servers view. This may happen if there are errors during startup.
    b. The Servers view should now display “webMethods Integration Server at localhost [Started, Synchronized].”
3. Right-click on “webMethods Integration Server at localhost[Started, Synchronized].”
4. Select “Add and Remove…” on the popup menu to open the Add and Remove… dialog.
5. Move the AddIntengersWebProject from the Available: list to the Configured: list. To move the project you can:
    a. Click the Add button.
    b. Or double click the AddIntengersWebProject in the Available list.
6. Make sure the “If server is started publish changes immediately” check box is selected, and then click the Finish button to publish the project.
    a. A “Publishing Project: AddIntengersWebProject” dialog should display and disappear quickly.
    b. If there are errors during publishing Designer should switch to the Console view to display the server errors.
7. If publishing is successful, you should see AddIntengersWebProject [Synchronized] displayed below “webMethods Integration Server at localhost[Started, Synchronized]” in the Servers view.

3.9 Accessing the Project on the Web

Access the AddIntegersWebProject application using a web browser.

1. Open any browser that you have installed on your local machine.
2. Type http://localhost:8072/AddIntegersWebProject/addIntegers.jsp to open you web client.

3. Start using the application.
    a. You can add two integers.
    b. And you can clear any data displayed on the web page.

3.10 Unpublishing the Project

Unpublish the project from Integration Server

1. Go to the Servers view.
2. Right-click on AddIntegersWebProject.
3. Select Remove from the popup menu that opens (alternatively you can use the Delete button on your keyboard).
4. Click OK on the Server confirmation dialog that opens.
5. An Unpublishing Project: AddIntengersWebProject dialog should display and disappear quickly.
6. And the AddIntengersWebProject [Synchronized] entry is removed from below “webMethods Integration Server at localhost[Started, Synchronized]” in the Servers view.

4 Update the Java Project to Access IS Services

This chapter expands on the first project (Ch. 3 above) and gives step-by-step instructions on how to create an Application Platform Java web project example that uses Java objects that you create to call IS services.

4.1 Prerequisites

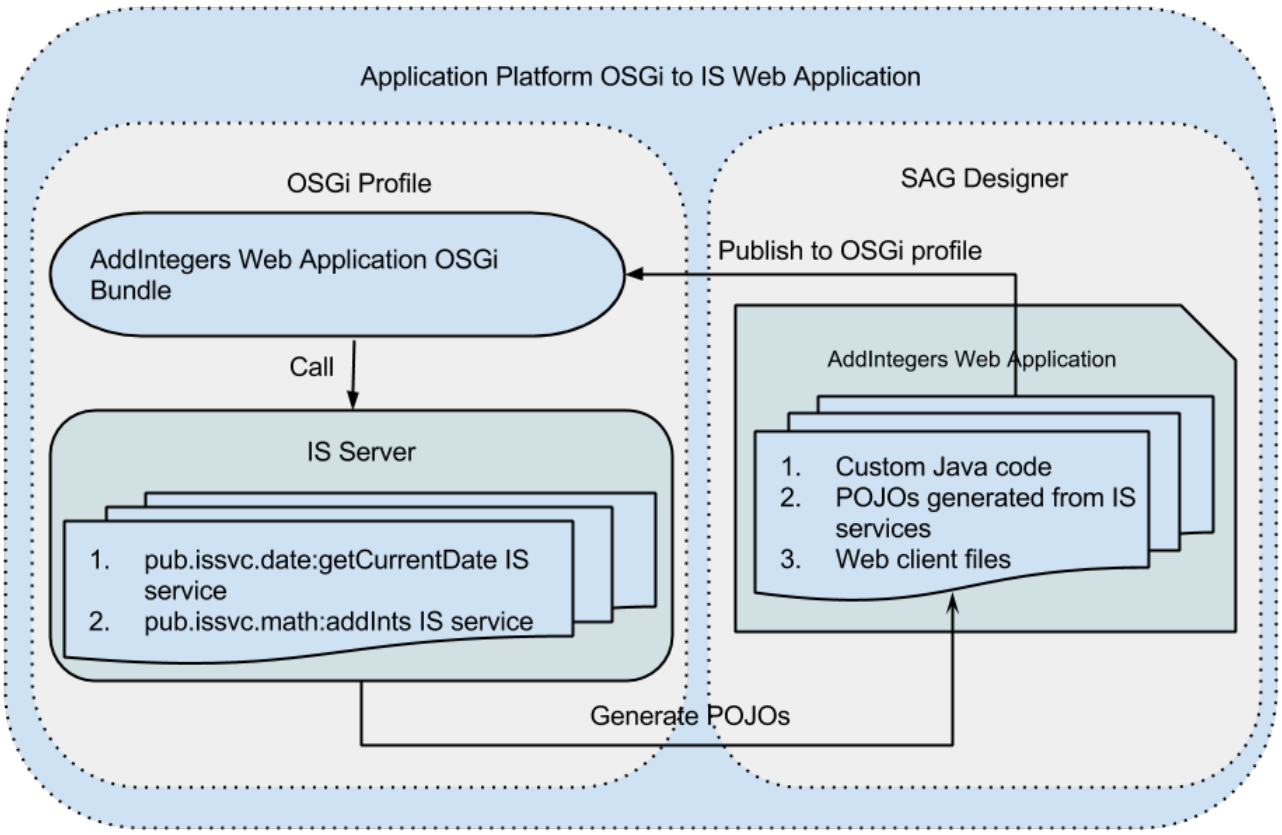
For this project it is assumed that:

- The IS package AppPlatformTutorial is installed in IS and available for use.
    - The package uses two IS flow services: pub.issvc.math:addInts and pub.issvc.date:getCurrentDate. These two flow services wrap IS public services: pub.math:addInts and pub.date:getCurrentDate which are needed for this example.

4.2 Architecture of the Example Application

The updated example application now consists of three main components: the AddIntegers web client and the AddIntegers OSGi Java application bundle, which gets deployed to the IS container's OSGi profile by the user when creating the AddIntegers web application - these two are as in the first project. You will now add the pub.issvc.math:addInts and pub.issvc.date:getCurrentDate IS flow services from package AppPlatformTutorial that should be installed in IS as a prerequisite..

Figure 4-1: Architecture of the updated AddIntegers web application, which is now an OSGi to IS web application



The AddIntegers web client is a JavaServer pages application that displays the current date on the application's webpage using the IS flow service pub.issvc.date:getCurrentDate; requests the user to enter two integers in two text fields, displayed on the page; and then adds the two numbers together using the pub.issvc.math:addInts IS flow service and displays the result of the addition to the user on a label on the web page.

The AddIntegers OSGi bundle contains the web client's web application files, such as JSP and HTML files, custom Java code created by the user and plain old Java objects (POJOs) generated from the pub.issvc.math:addInts and pub.issvc.date:getCurrentDate IS flow services.

pub.date:getCurrentDate and pub.math:addInts are preexisting public services found in the Integration Server installation and also accessible from the Designer Service Development perspective.

4.2.1 Tiers in the Example Application

The AddIntegersWebApplication has one web tier component (the AddIntegers web client) and three business tier components (the AddIntegers OSGi bundle and the pub.issvc.date:getCurrentDate and pub.issvc.math:addInts IS services). The user's web browser is the client tier component, as it accesses the rest of the application through the web.

4.3 Coding the Web Project Example

This section describes how to update the AddIntegers example web project created in the first project, so that it uses POJOs created from IS services.

4.4 Getting Started

Before you can proceed with this project, you must first perform the steps in chapter 3 above.

4.5 Integrating with Java Code Generated from IS service

The steps to follow for this are described in chapter 7 below. Make the following changes when following the steps defined in chapter 7:

1. Substitute AddIntegersWebProject for project
2. Select the following IS services AppPlatformTutorial.pub.issvc.date:getCurrentDate and AppPlatformTutorial.pub.issvc.math:addInts.

4.5.1 Coding against the generated Java code

You now need to edit the Java code under the src/main/java folder to use the generated POJO code.

1. Edit the AddIntegers.java class in the com.addints package so it looks as follows:

```
// Copyright (c) 2014-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc.,
Reston, VA, United States of America, and/or their licensors.
   // Use, reproduction, transfer, publication or disclosure is prohibited except as spe-
   cifically provided for in your License Agreement with Software AG.
   package com.addints;

   import pub.issvc.date.getcurrentdate.*;
   import pub.issvc.math.addints.*;

   public class AddIntegers {

      public java.util.Date getCurrentDate(){

           PLS_GetCurrentDate svc = new PLS_GetCurrentDate();
           PLS_GetCurrentDateOutput outputObject = null;
           try {
             //Does not expect any inputs
                outputObject = svc.invoke(null);
           }
           catch ( Exception e ) {
             e.printStackTrace();
           }
           // Get results from outputObject
             return outputObject.getDate();
      }
```

```
    public String addTwoIntegers(String num1, String num2){

        PLS_AddIntsInput inputObject = new PLS_AddIntsInput();
        //Populate inputObject
        inputObject.setNum1(num1);
        inputObject.setNum2(num2);
        PLS_AddInts svc = new PLS_AddInts();
        PLS_AddIntsOutput outputObject = null;
        try {
            outputObject = svc.invoke(inputObject);
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
        //Get results from outputObject
        //Only get the value of outputObject if it is not null
        //Initialize returnNum to an empty string instead of null because it looks nicer
         //when displayed in the GUI
        String returnNum="";
        if(outputObject!=null)
          returnNum = outputObject.getValue();

        return returnNum;
    }
}
```

4.6 Creating the Example Web Client

The web client code under the src/main/webapp folder should be exactly the same as for the first example project.

4.7 Compiling the Project

This is the same as in chapter 3.7 above:

4.8 Publishing the Project to Integration Server

This is the same as in chapter 3.8 above.

4.9 Accessing the Project on the Web

This is the same as in chapter 3.9 above.

4.10 Unpublishing the Project

This is the same as in chapter 3.10 above.

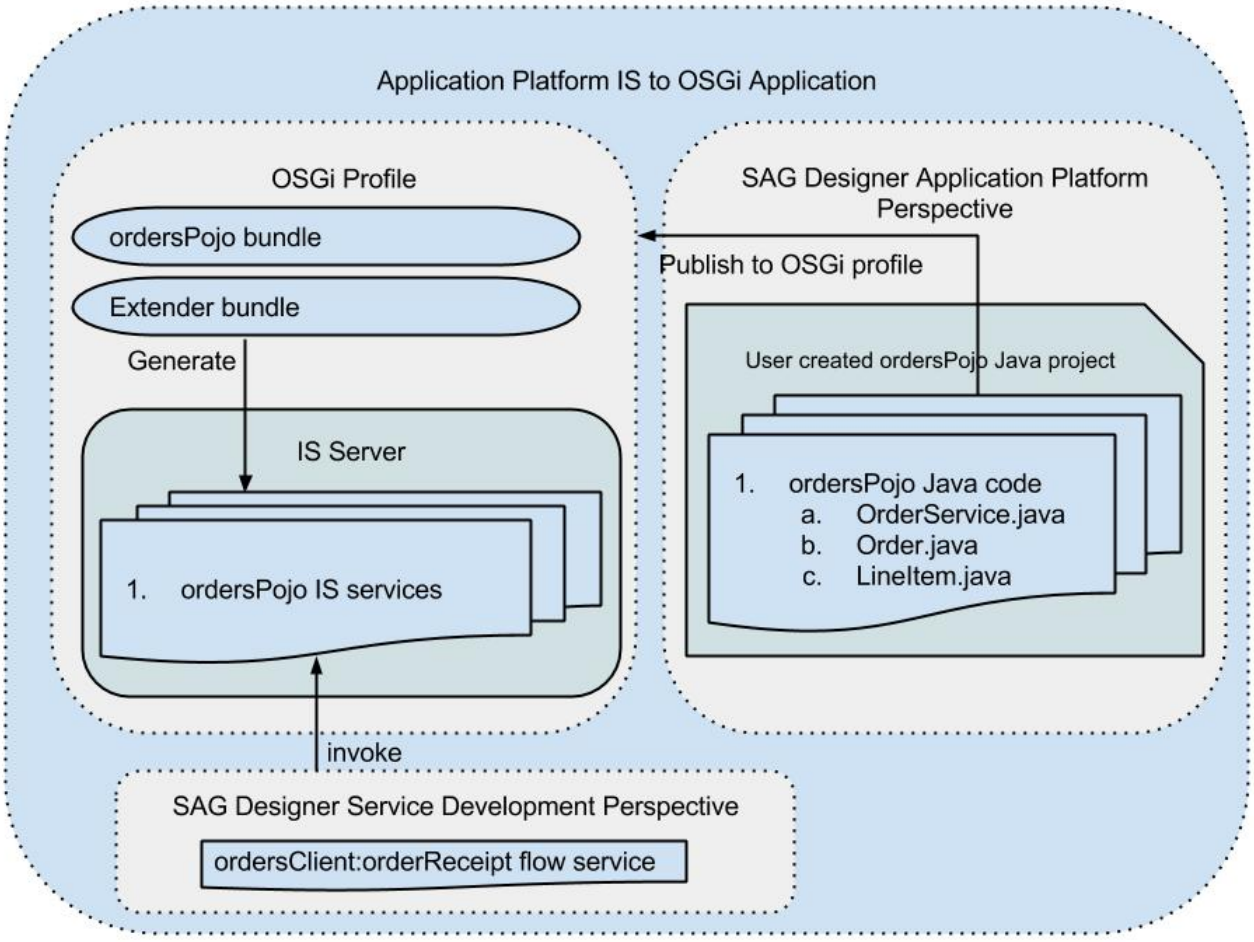
5 Creating a Third Java Project with IS Services Accessing Java Beans

This chapter gives step-by-step instructions on how to create an Application Platform Java project example that uses an IS Flow service to execute Java bean code that is deployed in IS as OSGI services.

5.1 Architecture of the Example Application

The example application has two main components: an ordersTestClient that has an orderReceipt IS flow service; and an ordersPojo project that has custom java code and is deployed as an OSGi bundle in the IS container's OSGi profile.

Figure 5-1: Architecture of the ordersPojo IS to OSGi Java project



ordersClient:orderReceipt is an IS Flow service that takes input values, "customer name" and two each of "item name", "item count" and "item price". It then assembles these into an Order with two line items and outputs the result as a receipt string that shows the order date, the input values entered by the user and the total cost of all the line items.

The flow service is executed from the Designer Service Development perspective.

The orderPojo project has the following custom Java code: LineItem.java, Orders.java, OrdersService.java and OrdersServiceImpl.java. OrdersService.java is an interface implemented by OrdersServiceImpl.java. OrdersService.java has methods for creating orders and line items, for adding line items to orders and for creating an order receipt. The ordersPojo project is created in Designer's Application Platform perspective using the Java Project wizard and is deployed to IS as an OSGi bundle.

5.1.1 Tiers in the Example Application

The ordersClient:orderReceipt Flow service is the client tier and can be invoked by the user from the Designer Service Development perspective or from any IS client. The business tier consists of the ordersPojo OSGi services.

5.2 Coding the Java Project Example

This section describes how to create the ordersPojo example java project.

5.3 Getting Started

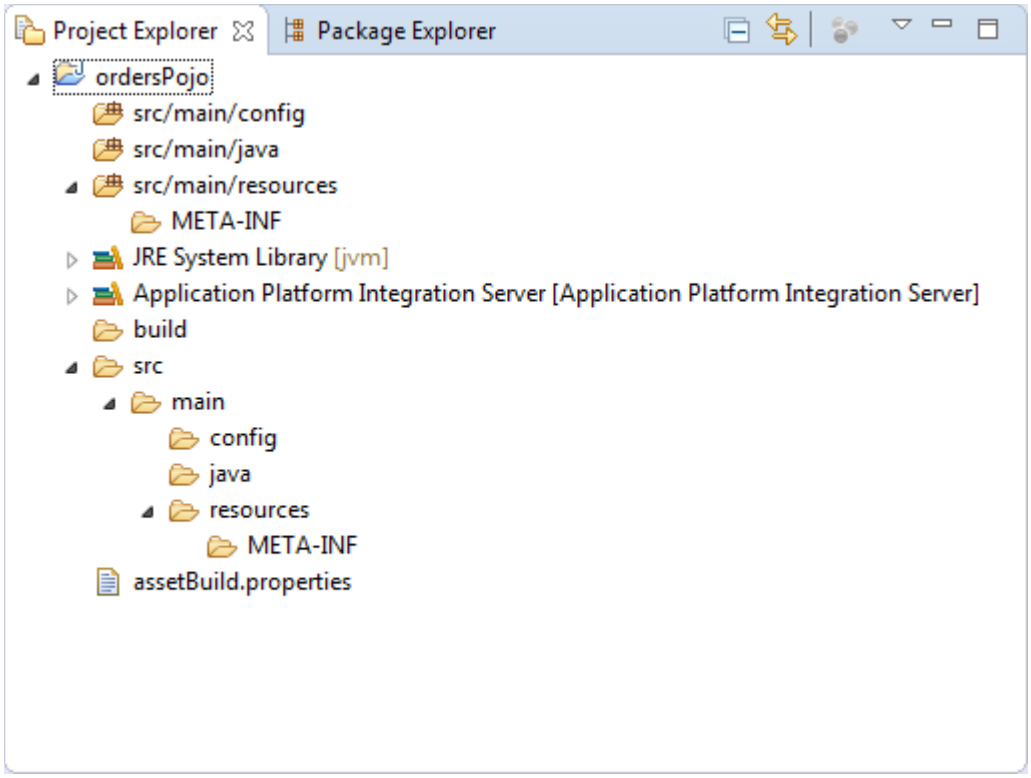
This section is the same as in chapter 3.3 above.

5.4 Creating the Java Project

1. In Designer close the Software AG welcome page if it is open - the Welcome page is displayed the first time you open a brand new Designer or if you create a new workspace.
2. In Designer's Window menu, choose Open Perspective.
3. Select Application Platform in the submenu that opens.
    a. If Application Platform is not available in the submenu, choose Other and then select Application Platform.
    b. An Application Platform perspective information dialog opens giving you a chance to configure an Application Platform runtime environment.
    c. Clicking the "Configure Application Platform runtime instance" will open the Server Runtime Environments page in the Preferences dialog.
    d. Click the Add... button to open the New Server Runtime Environment dialog and proceed as in 8.b below.
    e. Click OK in the Preferences dialog to close it when finished.
4. From the File menu, choose New and then Java Project to open the Application Platform Core Service Template dialog. Alternatively:
    a. From the File menu, choose New and then Other....
    b. Select Software AG > App Platform > Java Project in the new, "Select a wizard" dialog that opens and click Next to create a new project.

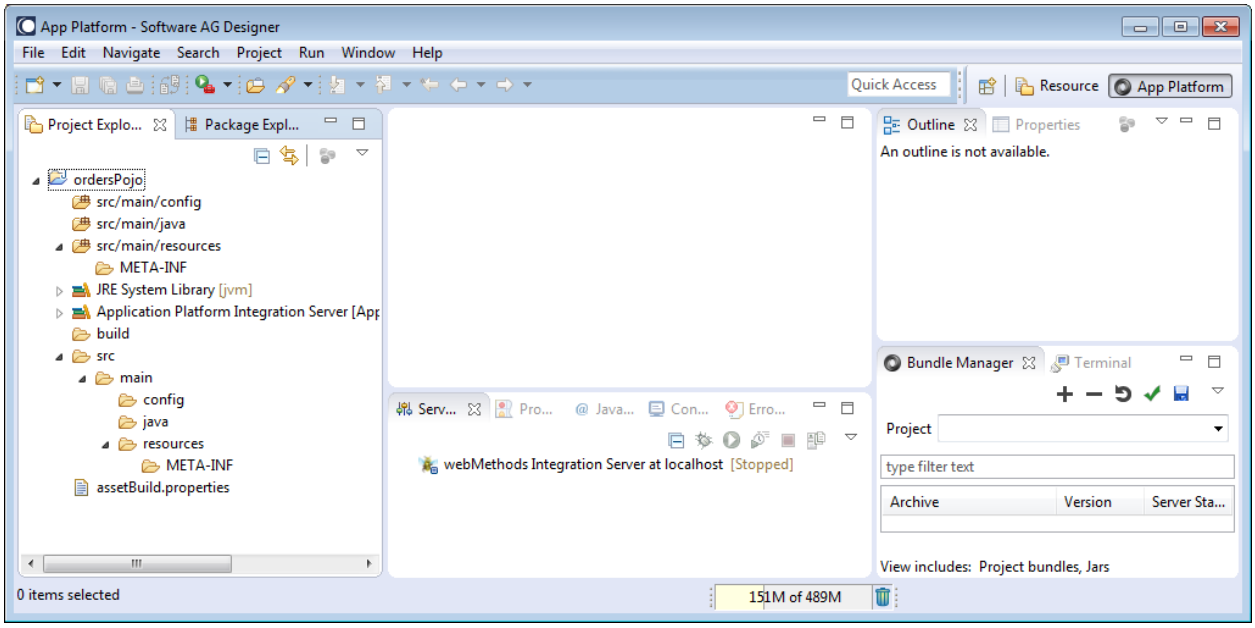
5. Enter a project name and click Next to go to the Project Facets page - *enter ordersPojo for our sample project*.
6. Confirm that Application Platform Core Service Preset is the selected configuration.
7. Verify that the following project facets are selected:
    a. Java
    b. Application Platform Core under SoftwareAG Application Platform
    c. Integration Server Extensions
8. Click on the Runtimes tab on the right hand of the dialog and select the “Application Platform Integration Server” checkbox if it is not selected.
    a. If the "Application Platform Integration Server" checkbox is not available, then click on the New... button to open the New Server Runtime Environment dialog.
    b. Expand the Software AG branch in the window and select "Application Platform Integration Server".
    c. Select the "Create a new local server" checkbox.
    d. Click the Next > button to go to the "New Application Platform Integration Server Runtime" page.
    e. Select the required JRE.
    f. Browse to select your IS Installation root directory.
    g. Click Next > to go to the "New Application Platform Integration Server Server" page.
    h. Make any required changes to the displayed property values if necessary otherwise use the default values.
    i. Click the Finish button to complete creating the new server runtime environment.
9. You could click Next > to accept or change the source directory name but for this tutorial can click the Finish button to create the project.
10. You should now see the project created in the Projects Explorer view and the project structure should be like the one in figure 5-2 below.

Figure 5-2: Sample view of the new ordersPojo Java project in Project Explorer



11. You should also see “webMethods Integration Server at localhost [Stopped]” added to the Servers view.

Figure 5-3: Sample view of the AppPlatform perspective just after creating the ordersPojo project displayed in the Project Explorer view and “webMethods Integration Server at localhost [Stopped]” added to the Servers view



5.4.1 Code your Java classes

You can now add your Java code under the src/main/java folder:

2. Expand src/main/java.
3. Right click, select New > Package to open the New Java Package dialog.
4. Type com.softwareag.demo.orders.api in the Name field.
5. Click Finish to create the package.
6. Create another package called com.softwareag.demo.orders.impl.
7. Right click the com.softwareag.demo.orders.api package and select New > Class to open the New Java Class dialog.
8. Type LineItem in the Name field.
9. Leave the rest of the values as default and click the Finish button to create the LineItem.java class.
10. The LineItem.java class should now be open in the Java editor in the Eclipse Workbench.
11. Type the following code in the editor:

```
// Copyright (c) 2014-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc.,
Reston, VA, United States of America, and/or their licensors.
// Use, reproduction, transfer, publication or disclosure is prohibited except as specifi-
cally provided for in your License Agreement with Software AG.
package com.softwareag.demo.orders.api;

/**
 * A Java bean to use in service method signature.
 */
public class LineItem {

    private String name;

    private int units;

    private float unitPrice;

    public LineItem() {
       this("dummy", -1, -1.0F);
    }

       public LineItem(String inName, int inUnits, float inPrice) {
       name = inName;
       units = inUnits;
       unitPrice = inPrice;
    }

    public String getName() {
            return name;
       }

       public void setName(String inName) {
```

```
            name = inName;
      }

      public int getUnits() {
            return units;
      }

      public void setUnits(int inUnits) {
            units = inUnits;
      }

      public float getUnitPrice() {
            return unitPrice;
      }

      public void setUnitPrice(float inPrice) {
            unitPrice = inPrice;
      }

    public String toString() {
        return "LineItem - " + units + " " + name + "s, at " + unitPrice + " each";
    }
}
```

12. Create another class Order.java in package com.softwareag.demo.orders.api.
13. Type the following code in Order.java:

```
// Copyright (c) 2014-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc.,
Reston, VA, United States of America, and/or their licensors.
// Use, reproduction, transfer, publication or disclosure is prohibited except as specifi-
cally provided for in your License Agreement with Software AG.
package com.softwareag.demo.orders.api;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.List;

import static java.lang.System.lineSeparator;

/**
 * A Java bean to use in service method signature.
 */
public class Order {

    String customerName;

    Date orderDate;

    List<LineItem> lineItems = new ArrayList<>();

    public Order() {
      this("dummy", new Date());
    }
```

```
    public Order(String inName) {
      this(inName, new Date());
    }

    public Order(String inName, Date inDate) {
      setCustomerName(inName);
      setOrderDate(inDate);
    }

    public String getCustomerName() {
            return customerName;
      }

      public void setCustomerName(String inName) {
            customerName = inName;
      }

      public Date getOrderDate() {
            return orderDate;
      }

      public void setOrderDate(Date inDate) {
            orderDate = inDate;
      }

    public int addLineItem(LineItem inItem) {
        lineItems.add(inItem);
        return lineItems.size();
    }

    public List<LineItem> getLineItems() {
        return new ArrayList<>(lineItems);
    }

    public void setLineItems(List<LineItem> inList) {
      lineItems = ( inList != null && inList.size() > 0 ) ? new ArrayList<>(inList) : new
ArrayList<LineItem>();
    }

      public String toString() {
      StringBuilder buff = new StringBuilder();
        buff.append("Order - ").append(customerName).append(" on ");
        buff.append(orderDate).append("), items:");
        if ( notEmpty( lineItems )  ) {
            for ( LineItem nextItem : lineItems ) {
                buff.append(lineSeparator()).append(nextItem.toString());
            }
        }
        else {
            buff.append(" NONE").append(lineSeparator());
        }
        return buff.toString();
    }

    private static boolean notEmpty(Collection<?> inCollection) {
```

```
        return inCollection != null && inCollection.size() > 0;
    }
}
```

14. Right click the com.softwareag.demo.orders.api package and select New > Interface to open the New Java Interface dialog.
15. Type OrdersService in the Name field.
16. Leave the rest of the values as default and click the Finish button to create the OrdersService.java interface.
17. Type the following code in OrdersService.java:

```
// Copyright (c) 2014-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc.,
Reston, VA, United States of America, and/or their licensors.
// Use, reproduction, transfer, publication or disclosure is prohibited except as specifi-
cally provided for in your License Agreement with Software AG.
package com.softwareag.demo.orders.api;

/**
 * Interface for a simple Orders OSGi service.
 */
public interface OrdersService {

      /**
     * Create receipt fragment for the input item.
     */
    public String createReceiptEntry(LineItem inItem);

    /**
     * Create receipt fragment for the input order.
     */
    public String createReceipt(Order inOrder);

    /**
     * Calculate charge for the input item.
     */
    public float calculateCharge(LineItem inItem);

    /**
     * Create a new line item.
     */
    public LineItem createLineItem(String inName, int inCount, float inPrice);

    /**
     * Create a new order.
     */
    public Order createOrder(String inCustomer);

    /**
     * Add the input new line item to the input order.
     */
    public Order addLineItem(Order inOrder, LineItem inItem);
}
```

18. Create OrdersServiceImpl.java in package com.softwareag.demo.orders.impl.

19. Type the following code in OrdersServiceImpl.java:

```
// Copyright (c) 2014-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc.,
Reston, VA, United States of America, and/or their licensors.
// Use, reproduction, transfer, publication or disclosure is prohibited except as specifi-
cally provided for in your License Agreement with Software AG.
package com.softwareag.demo.orders.impl;

import java.util.List;

import static java.lang.System.lineSeparator;

import com.softwareag.applatform.sdk.annotations.ExposeToIS;
import com.softwareag.applatform.sdk.annotations.ExposedMethod;
import com.softwareag.applatform.sdk.annotations.Service;
import com.softwareag.demo.orders.api.LineItem;
import com.softwareag.demo.orders.api.Order;
import com.softwareag.demo.orders.api.OrdersService;

/**
 * Implementation for the simple Orders OSGi service.
 */
@Service(name="OrdersService", interfaces={"com.softwareag.demo.orders.api.OrdersService"})
@ExposeToIS(packageName="OrdersService")
public class OrdersServiceImpl implements OrdersService {

    /**
     * Create receipt fragment for the input item.
     */
    @Override
    @ExposedMethod
    public float calculateCharge(LineItem inItem) {
        return ( inItem == null ) ?  0.0F : inItem.getUnits() * inItem.getUnitPrice();
    }

    /**
     * Calculate charge for the input item.
     */
    @Override
    @ExposedMethod
    public String createReceiptEntry(LineItem inItem) {
      StringBuilder buff = new StringBuilder();
        if ( inItem == null  ) {
            buff.append("  NULL Line Item").append(lineSeparator());
        }
        else {
            buff.append("  ").append(inItem.getName());
            buff.append(": ").append(inItem.getUnits());
            buff.append(" @ ").append(inItem.getUnitPrice()).append(" comes to ");
            buff.append(calculateCharge(inItem));
        }
```

```
        return buff.toString();
    }

    /**
     * Create receipt fragment for the input order.
     */
    @Override
    @ExposedMethod
    public String createReceipt(Order inOrder) {
      StringBuilder buff = new StringBuilder();
        if ( inOrder == null  ) {
            buff.append("Cannot produce receipt for NULL Order").append(lineSeparator());
        }
        else {
            buff.append(inOrder.getCustomerName()).append(" on
").append(inOrder.getOrderDate());
            buff.append(lineSeparator()).append( "items:" );
            List<LineItem> items = inOrder.getLineItems();
            if ( items.size() == 0 ) {
                buff.append(" NONE").append(lineSeparator());
            }
            else {
             float total = 0.0f;
                for ( LineItem nextItem : items ) {
                   total += getSubtotal( nextItem );
                    buff.append(lineSeparator()).append( createReceiptEntry(nextItem) );
                }
                buff.append(lineSeparator()).append( createReceiptTotalLine(total) );
            }
        }
        return buff.toString();
    }

    private static float getSubtotal(LineItem inItem) {
        return ( inItem == null ) ? 0.0f : inItem.getUnits() * inItem.getUnitPrice();
   }

    private static String createReceiptTotalLine(float inTotal) {
        return String.format( "  Total: $%8.2f", inTotal );
   }

    /**
     * Create a new line item.
     */
    @Override
    @ExposedMethod
    public LineItem createLineItem(String inName, int inCount, float inPrice) {
        return new LineItem(inName, inCount, inPrice);
   }

    /**
     * Create a new order.
     */
    @Override
    @ExposedMethod
    public Order createOrder(String inCustomer) {
```

```
        return new Order(inCustomer);
    }

    /**
     * Add the input new line item to the input order.
     */
    @Override
    @ExposedMethod
    public Order addLineItem(Order inOrder, LineItem inItem) {
        inOrder.addLineItem(inItem);
        return inOrder;
    }
}
```

5.5 Compiling the Project

This section is the same as in chapter 3.7 above. The only difference is that you should select the ordersPojo checkbox if you only want to build the orderPojo project.

5.6 Publishing the Project to Integration Server

This section is the same as in chapter 3.8 above. The only difference is that you should choose ordersPojo for publishing.

5.7 Create the Test Client Flow Service.

1. Switch to the Service Development perspective.
    a. If it is not visible on the toolbar then:
        i. In Designer's Window menu, choose Open Perspective.
        ii. Select Service Development in the submenu that opens.
        iii. If Service Development is not available in the submenu, choose Other, and then select Service Development.
2. Verify that your new ordersPojo Java code has been converted into IS services by confirming that:
    a. There is an ordersPojo package with the following services under com.softwareag.demo.orders.api.OrdersService:
        i. addLineItem
        ii. calculateCharge
        iii. createLineItem
        iv. createOrder
        v. createReceipt
        vi. createReceiptEntry

3. Copy the OrdersTestClient.zip file to the IS replication/inbound directory and import it into your IS.
4. From Designer's Service Development perspective find the OrdersTestClient package and then find the orderReceipt flow service in the ordersClient folder.

5.8 Running the Test Client Flow Service

Run the 'ordersClient:orderReceipt' Flow service from the Designer Service Development perspective.

4. Right-click orderReceipt, click on “Run As,” and select “Run Flow Service” to open the “Enter Input for ‘orderReceipt’” dialog.
5. Enter values for the inputs as shown in figure 5-4 below. All inputs accept strings.
6. Click OK to run the service.
7. The result is displayed and the output fields are a record representing the final Order object and a receipt string. The result should be similar to that displayed in figure 5-5 below.

Figure 5-4: Showing the Flow runtime input dialog as seen in Designer’s Service Development perspective

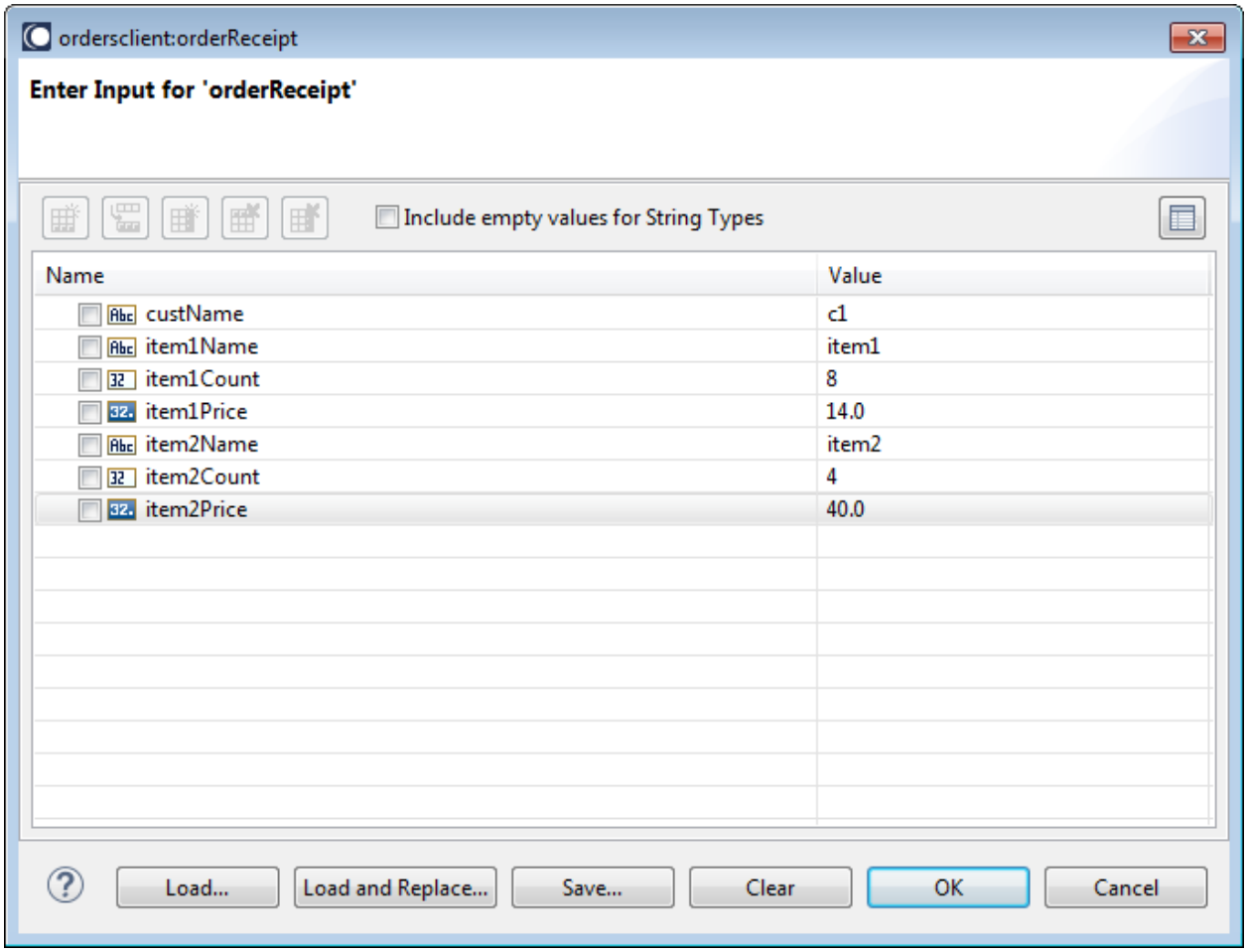
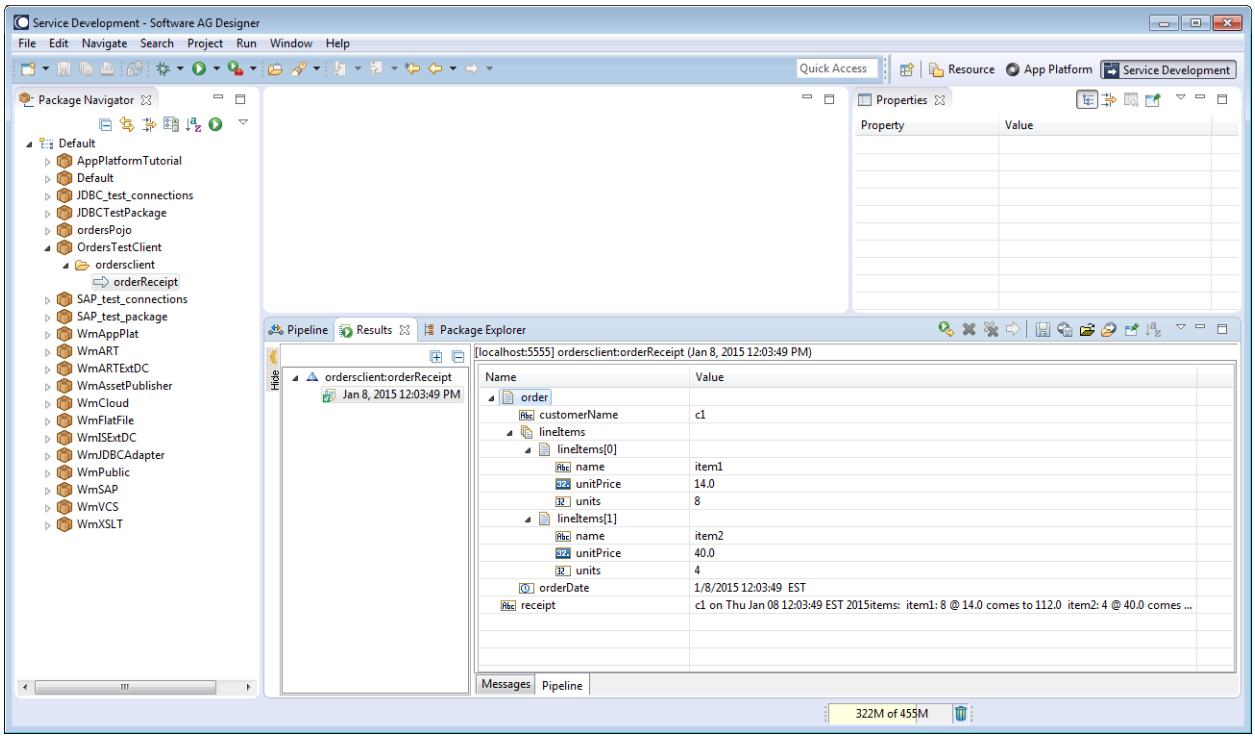


Figure 5-5: Designer Service perspective showing the results of the ordersPojoTestClient execution



5.9 Unpublishing the Project

This section is the same as in chapter 3.10 above. The only difference is to choose ordersPojo for unpublishing.

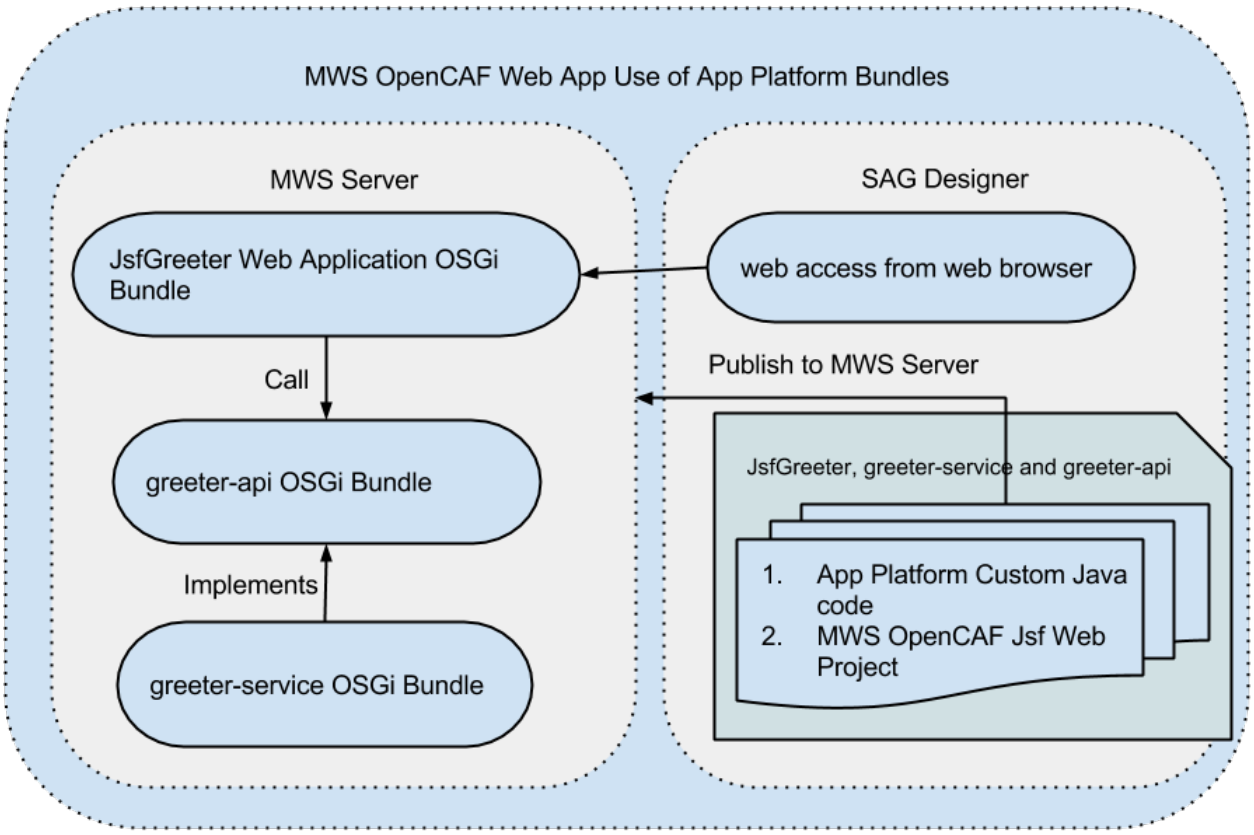
6 Create an MWS CAF Web Project That Uses Pre-existing Application Platform Projects

This chapter gives step-by-step instructions on how to create an MWS CAF web project that makes use of services in pre-existing Application Platform Java projects.

6.1 Architecture of the Example Application

This example application has three projects - two Application Platform projects (greeter-api and greeter-service) and an MWS CAF web project (JsfGreeterWeb). The greeter-service project has two Java classes called GreeterImpl.java and ResourceUtil.java. GreeterImpl.java is an implementation of IGreeter.java, which can be found in the greeter-api project. ResourceUtil.java is for accessing message resource files under src/main/resources folder in the greeter-service project. Both greeter-api and greeter-service projects are deployed as OSGi bundles to the following location, "(install_root)\profiles\(MWS_instance)\workspace\app-platform\deployer\bundles" and JsfGreeterWeb project is deployed as an OSGI bundle to "(install_root) \MWS\server\default\deploy" folder.

Figure 6-1: Architecture of the greeter web application



6.1.1 Tiers in the Example Application

JsfGreeter web client is the web tier component for this web application.The greeter-api and greeter-service OSGi bundles make up the business tier. The user's web browser is the client tier component.

6.2 Coding the Web Project Example

This section describes how to create the JsfGreeter example web project.

6.3 Getting Started

Before you can start coding the example, you need to perform the following steps:

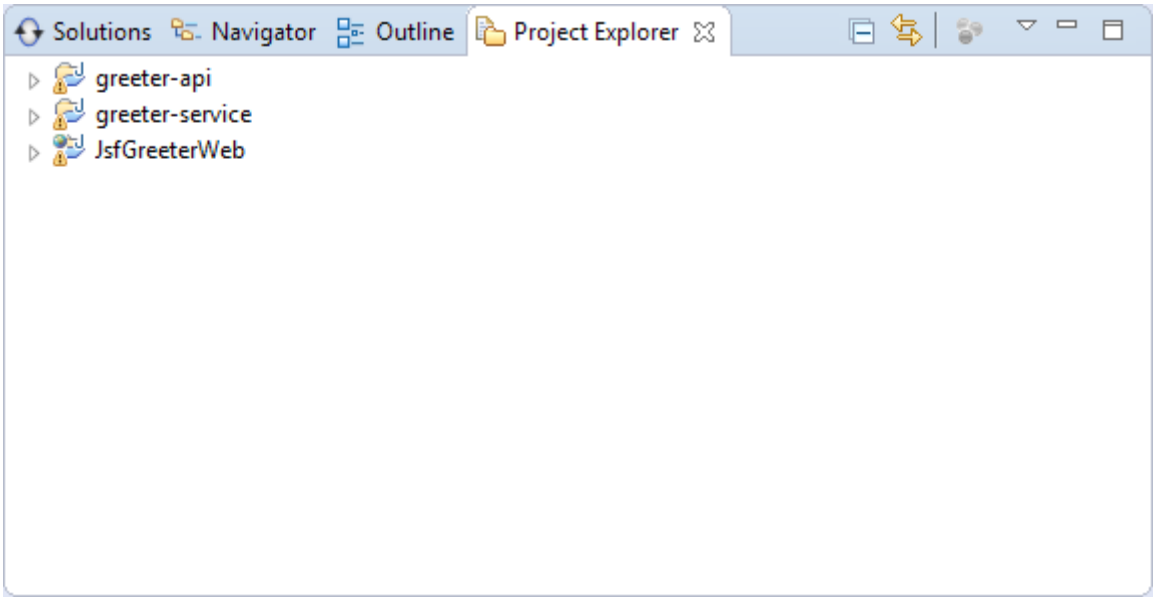
1. Install Application Platform Development typical installation
2. Start Software AG Designer

6.4 Creating the Project

1. In the Software AG welcome page in Designer select the “UI Development perspective” - the Welcome page is displayed the first time you open a brand new Designer or if you create a new workspace.
2. Login as sysadmin/manage if/when the "MWS Server Credentials" dialog pops up for localhost:5002
3. Import the greeter-api, greeter-service and JsfGreeterWeb projects into Designer.
    a. The projects can be downloaded from the TechCommunity website from the URL http://techcommunity.softwareag.com/pwiki/-/wiki/Main/Application+Platform.
    b. Use the Software AG => “Existing CAF Projects into Workspace” import method for all three projects because the Software AG Designer import wizard automatically repairs the project while the other import wizards do not.
4. When finished you should see the projects in the Projects Explorer view as follows

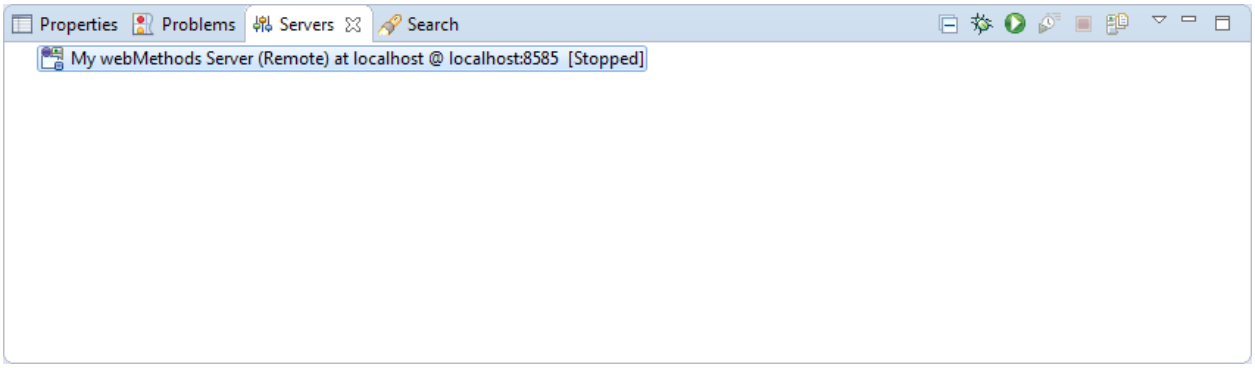
    Note: If you want to create a JsfGreeterWeb CAF project of you own follow the instructions in the “CAF Development Help” guide.

Figure 6-2: Sample view of the JsfGreeterWeb project and related imported greeter-api and greeter-service projects in Project Explorer



5. Confirm that you are in the UI Development perspective and that you see “My webMethods Server (Remote) at localhost [Stopped]” in the Servers view as shown below.

Figure 6-3: Sample Servers view showing “My webMethods Server (Remote) localhost @ localhost:8585 [Stopped]” while in the UI Development perspective



6.5 Add custom Java code to GreeterBean.java class

1. Expand Java Resources/src/ caf.war.JsfGreeterWeb
2. Open the GreeterBean.java class and you will see it imports com.softwareag.applatform.sdk.ServiceUtil. This is part of the App Platform SDK. The SDK is added to the classpath using the new API classpath container that is used to lookup App Platform published services. To access the services from the greeter-api bundle the GreeterBean class does a call-through as shown in the descibeGreeter method.

```
package caf.war.JsfGreeterWeb;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

import com.example.osgi.greet.api.IGreeter;
import com.softwareag.applatform.sdk.ServiceUtil;
import com.webmethods.caf.faces.annotations.BeanType;
import com.webmethods.caf.faces.annotations.DTManagedBean;
import com.webmethods.caf.faces.annotations.ExpireWithPageFlow;

@ManagedBean(name = "GreeterBean")
@SessionScoped
@ExpireWithPageFlow
@DTManagedBean(beanType = BeanType.DEFAULT)
public class GreeterBean extends com.webmethods.caf.faces.bean.BaseFacesSessionBean {

	private java.lang.String myGreeter;

	public GreeterBean()
	}

	public String describeGreeter() {
		System.err.println("Invoking GreeterBean#describeGreeter()");
		IGreeter greeter = ServiceUtil.getService(IGreeter.class);
		return greeter.describeGreeter();
	}
	public String getGreeter() {
		return describeGreeter();
```

```
        }
    public java.lang.String getMyGreeter()  {

        return myGreeter;
    }
    public void setMyGreeter(java.lang.String myGreeter)  {
        this.myGreeter = myGreeter;
    }

    /**
   * Override this method to release any resources associated with this session.
   * Please note, the FacesContext is not valid for this function
   */
  protected void release() { }

    }
```

3. Service project GreeterImpl class has the following method:

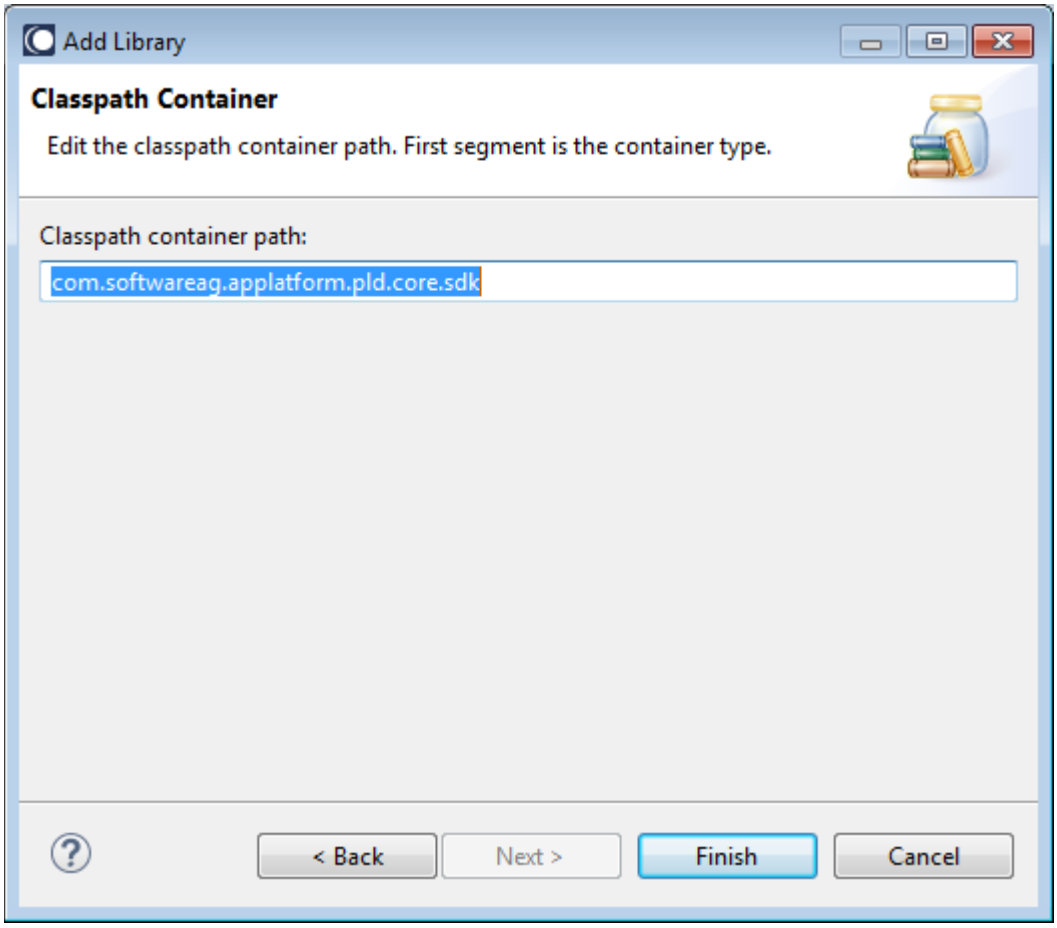
```
    @Override
    public String describeGreeter() {
        System.err.println("Invoking GreeterImpl#describeGreeter() service
project...");
        if ( name == null) {
            return getSessionUser();
        }
        return name;
    }
```

6.6 How to Use the API Classpath Container

Add the com.softwareag.applatform.sdk API to classpath as follows:

1. Right-click on the JsfGreeterWeb project and select Properties to open the “Properties for JsfGreeterWeb” dialog.
2. Select “Java Build Path” and then select the Libraries tab in the “Properties for JsfGreeter-Web” dialog.
3. Click on the “Add Library…” button to open the standard “Add Library” dialog.
4. Choose “Application Platform API Bundles”. You should see the following dialog.

   Figure 6-4: Application Platform API Bundles, “Add Library” dialog



5. Click the Finish button to close the dialog and add the API libraries to the build path.
6. Click OK to close the "Properties for JsfGreeterWeb" dialog.

6.7 Compile the Project

Designer can be set to build projects automatically. You can also build projects manually. If you are compiling manually:

1. In Designer's Project menu select the "Build Automatically" menu item.
2. Choose Clean… in the same Project menu to open the Clean dialog.
3. Choose the "Clean projects selected below" radio button and select only the AddIntengersWebProject checkbox to only build this project.
4. Click OK to build the project.

6.8 Publishing the Project to My webMethods Server

Publish the projects to the MWS Server.

1. Go to the Servers view.

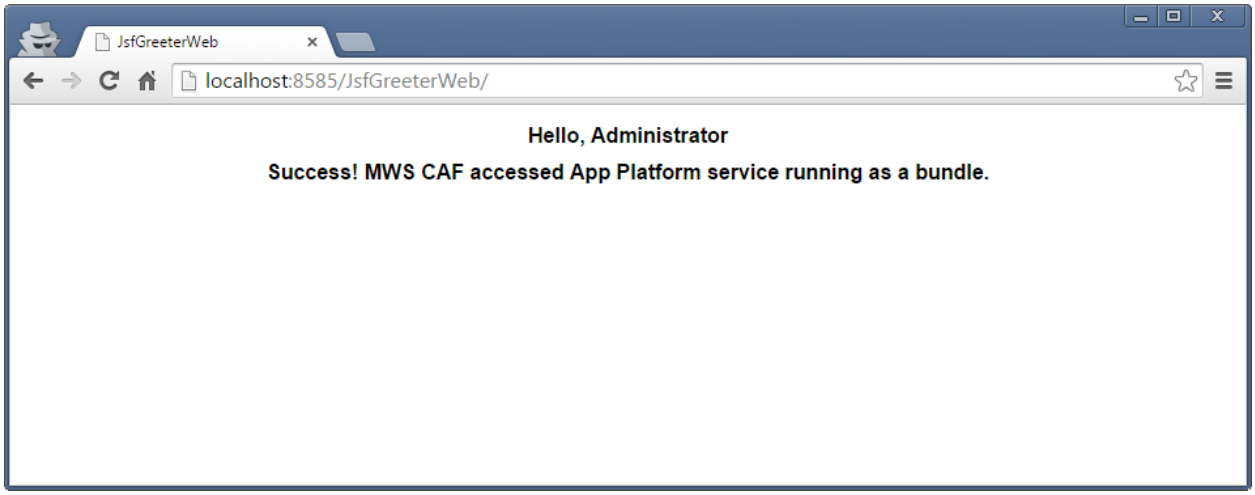
2. If the server is not already started, start it by right clicking on “My webMethods Server (Remote) at localhost @ localhost:8585 [Stopped]” and clicking Start on the popup menu.
    a. When started, the Servers view will show “My webMethods Server (Remote) at localhost @ localhost:8585 [Started, Synchronized].”
    b. Note: "My webMethods Server (Remote) at localhost @ localhost:8585 [Stopped]" can vary depending on how the server name has been configured.
3. Right-click on “My webMethods Server (Remote) at localhost @ localhost:8585 [Started, Synchronized].”
4. Select “Add and Remove…” on the popup menu to open the Add and Remove… dialog.
5. Move the greeter-api, greeter-service and JsfGreeterWeb from the Available: list to the Configured: list. To move each project, you can:
    a. Click the Add button.
    b. Or double-click a project in the Available: list.
6. Make sure the “If server is started, publish changes immediately” check box is selected, and then click the Finish button to publish the project.
    a. A “Publishing Project…” dialog should display and disappear quickly.
    b. If there are errors during publishing, Designer should switch to the Console view to display the server errors.
7. An "MWS Server Credentials" dialog may pop up for localhost:5002 when publishing the JsfGreeterWeb project - usually if it is the first time publishing it. If that is the case, enter sysadmin/manage as credentials.
8. If publishing is successful, you should see all [Synchronized] next to each project below “My webMethods Server (Remote) at localhost @ localhost:8585 [Started, Synchronized]” in the Servers view.

6.9 Accessing the Project on the Web

Access the JsfGreeterWeb application from a browser as follows:

1. Open any browser that you have installed on your local machine.
2. Type http://localhost:8585/JsfGreeterWeb to open the web client.
3. The page requests a login.
4. Enter ‘Administrator/manage’ for credentials.The browser will transfer to a page that greets Administrator as “Hello Administrator” - it should look like the following figure:

Figure 6-5: Shows the JsfGreeter web application running in a browser.



6.10 Unpublishing the Project

Unpublish the project from Integration Server.

1. Go to the Servers view.
2. Right-click on each project.
3. Select Remove from the popup menu that opens (alternatively, you can use the Delete button on your keyboard).
4. Click OK on the Server confirmation dialog that opens.
5. An “Unpublishing Project…” dialog should display and disappear quickly.
6. Everything is removed from below “My webMethods Server (Remote) at localhost @ localhost:8585 [Started, Synchronized]” in the Servers view.

7 Using Bundle Manager to Manage Bundle Dependencies

Application Platform allows users to create projects that have third-party bundle dependencies and deploy them to a webMethods Integration Server container. This is done using the Application Platform Bundle Manager. This chapter gives step-by-step instructions on how to use the Bundle Manager. For this example you need to import the sample1 project into Designer.

7.1 Prerequisites

The following jars are required for this project (they are included as part of this tutorial):

1. sample1.zip
2. dm_sample_extension.jar

Extract the contents of sample1.zip to your file system. This will create the folder sample1.

7.2 Architecture of the Example Application

The sample1 example application has three main components: the ShowName Java class, the dm_sample_extension.jar third-party library, and a dm_sample_extension_1.0.0.jar OSGI wrapper bundle. The dm_sample_extension.jar third-party library contains Java classes and interfaces, required by the sample1 project. The dm_sample_extension_1.0.0.jar OSGI wrapper bundle wraps the dm_sample_extension.jar, is required at runtime by the sample1 application, and is deployed to the IS container's OSGi profile along with the sample1 application.

7.3 Import the Sample Project into Eclipse

This section describes how to setup the sample1 example project.

7.4 Getting Started

This section is the same as in chapter 3.3 above.

7.5 Creating the Project

1. In Designer close the Software AG welcome page if it is open - this displays the first time you open a brand new Designer or when you create a new workspace.
2. In Designer's Window menu, choose Open Perspective.
3. Select Application Platform in the submenu that opens.
    a. If Application Platform is not available in the submenu, choose Other and then select Application Platform.

b. An Application Platform perspective information dialog opens, allowing you to configure an Application Platform runtime environment.

c. Clicking the Yes button will open the Server Runtime Environments page in the Preferences dialog.

d. Click the Add... button to open the New Server Runtime Environment dialog and proceed as in 12 b below.

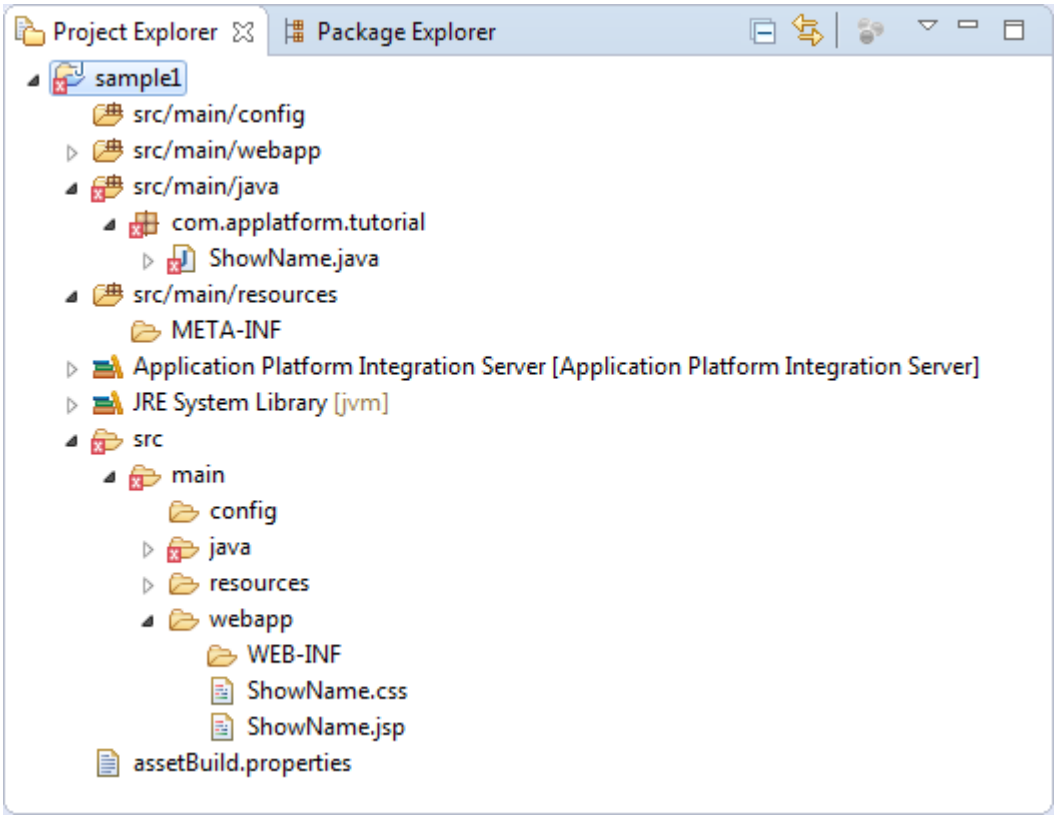
e. Click OK in the Preferences dialog to close it when finished and open the App Platform perspective.

4. From the File menu, choose Import... to open the Import dialog.
5. Select General > "Existing Projects into Workspace" and click Next > to go to the "Import Projects" page.
6. Browse to the folder where you extracted the sample1 project and select the root folder.
7. Select "Copy projects into workspace."
8. Click Finish to import the project into Designer. You should now see the project created in the Projects Explorer view and it should look like the one in figure 7-1 below.

    a. The compilation errors you see are expected when you first import the project.

    b. The spring folder is required because this is a Spring project.

Figure 7-1: The new sample1 project as seen in Project Explorer - compilation errors are expected at this point.



9. Right-click the sample1 project and select Properties to open the “Properties for sample1” dialog.
10. Select Application Platform > Project Bundle and check that the Web Context Path is sample1 - if not change it.
11. Select Project Facets on the tree menu on the left to open the Project Facets page.
12. Confirm that the following project facets are preselected for you:
    a. Java
    b. Application Platform Core under SoftwareAG Application Platform
    c. Application Platform Web also under SoftwareAG Application Platform
13. Confirm Integration Server Extensions is unchecked, which it should be by default.
    Note: we will not generate any Java code from IS services for this example so Integration Server Extensions is not needed.
14. Click on the Runtimes tab on the right hand of the dialog and select the “Application Platform Integration Server” check box if it is not selected.
    a. If the "Application Platform Integration Server" checkbox is not available then click on the New... button to open the New Server Runtime Environment dialog.
    b. Expand the Software AG branch in the window and select "Application Platform Integration Server."
    c. Select the "Create a new local server" checkbox.
    d. Click the Next > button to go to the "New Application Platform Integration Server Runtime" page.
    e. Select the required JRE.
    f. Browse to select your IS Installation root directory.
    g. Click Next > to go to the "New Application Platform Integration Server Server" page.
    h. Make any required changes to the displayed property values, if necessary. Otherwise use the default values.
    i. Click the Finish button to complete creating the new server runtime environment.
15. Click the Ok (or Apply) button to close the dialog and apply your changes.

Note: It is expected that you will see compilation errors at this point which will be fixed shortly using Bundle Manager and Application Platform Shared Bundles tools.

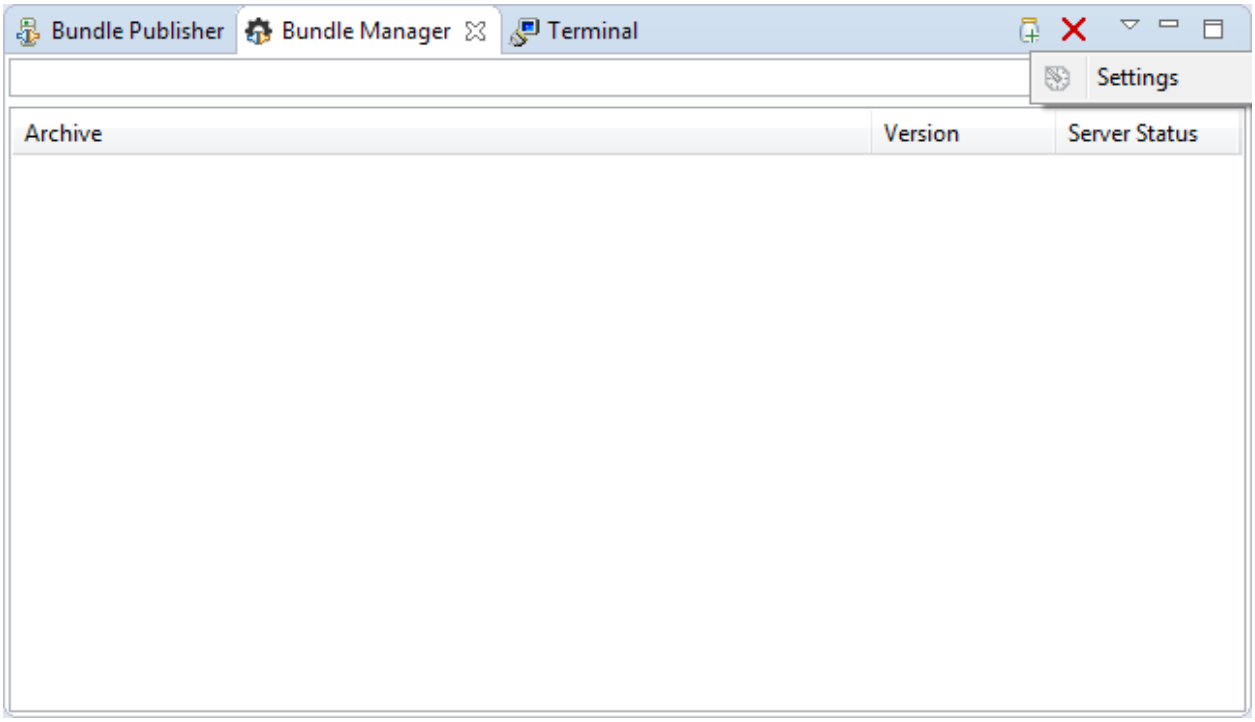
7.6 Using Bundle Manager to Create a Wrapper Bundle

The Application Platform Bundle Manager is an Eclipse view included in the Designer Application Platform perspective and is used to manage dependencies to other bundles. We will use it to create a wrapper bundle for the dm_sample_extension.jar needed for this example project included as part of this tutorial.

If Bundle Manager is not already visible in the Designer Application Platform perspective, which it should be by default, add it by navigating to the "Window/Show View/Other.../Software AG Applica-

tion Platform" menu path and selecting Bundle Manager to open the Bundle Manager view, as shown in figure 7-2 below.

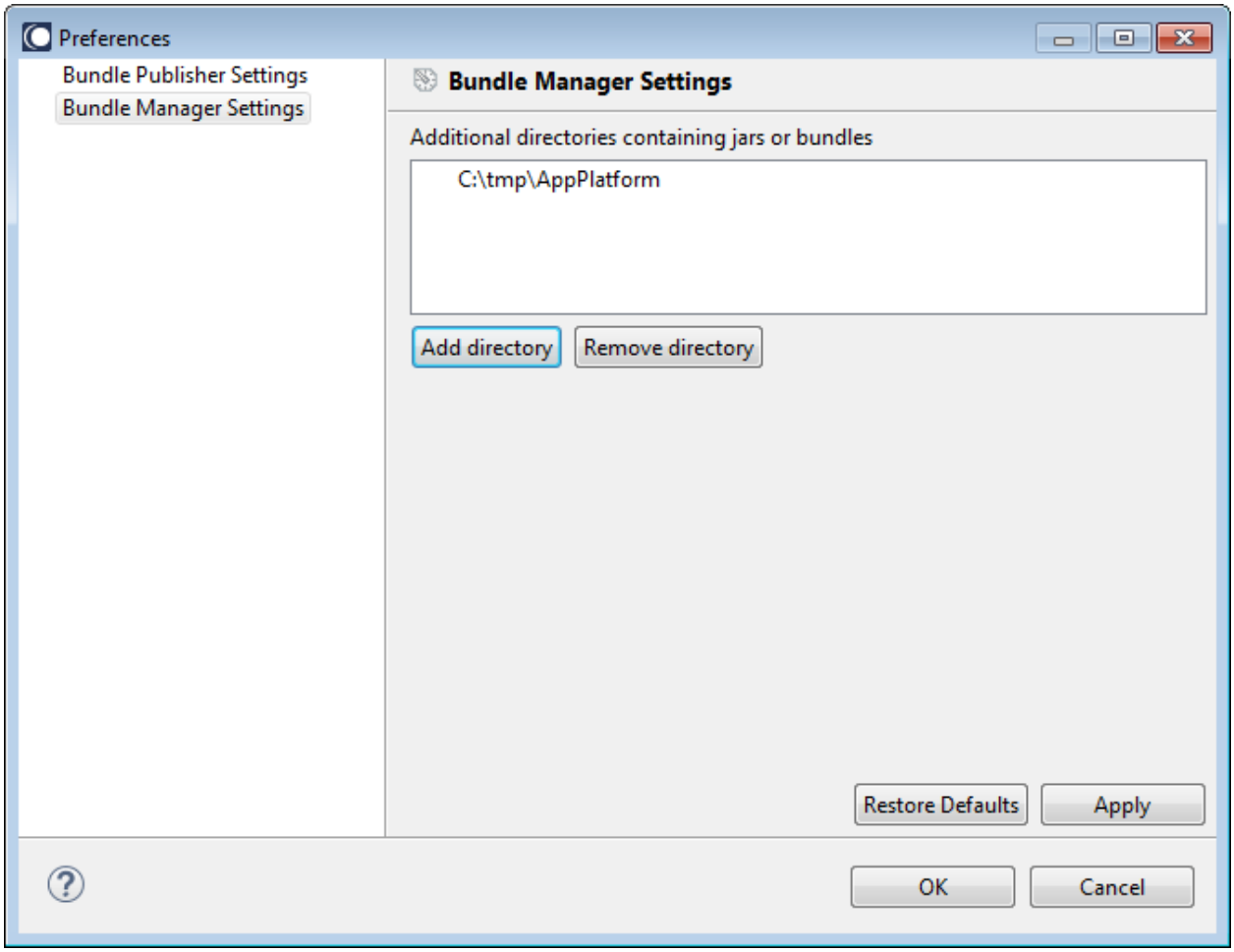
Figure 7-2: The Bundle Manager view - here it is shown displaying the Settings menu after the user has clicked on the “View Menu” icon, which is third on the toolbar



To create a wrapper bundle perform the following steps:

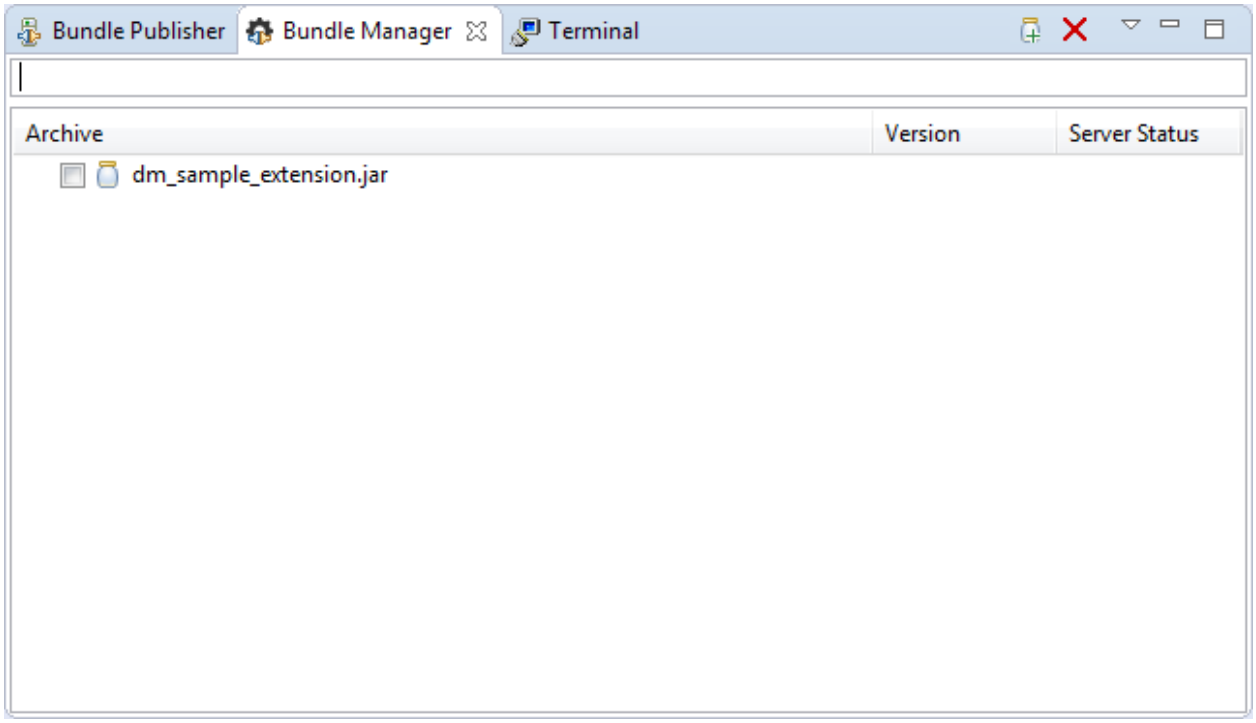
1. Click on the third icon on Bundle Manager’s toolbar, located at the far top right of Bundle Manger, to display the settings menu as shown in figure 7-2 above
2. Click on the Settings menu to open the Preferences dialog.
3. Make sure that the Bundle Manager Settings page is selected as shown in figure 7-3 below.
4. Click the “Add directory” button to add the directory where the dm_sample_extension.jar is located. After the directory is added the settings page will look as shown in figure 7-3 below.

Figure 7-3: Preferences dialog showing the Bundle Manager Settings page, which is displayed when the Bundle Manager Settings item is selected on the left hand side list. It also shows an added directory.



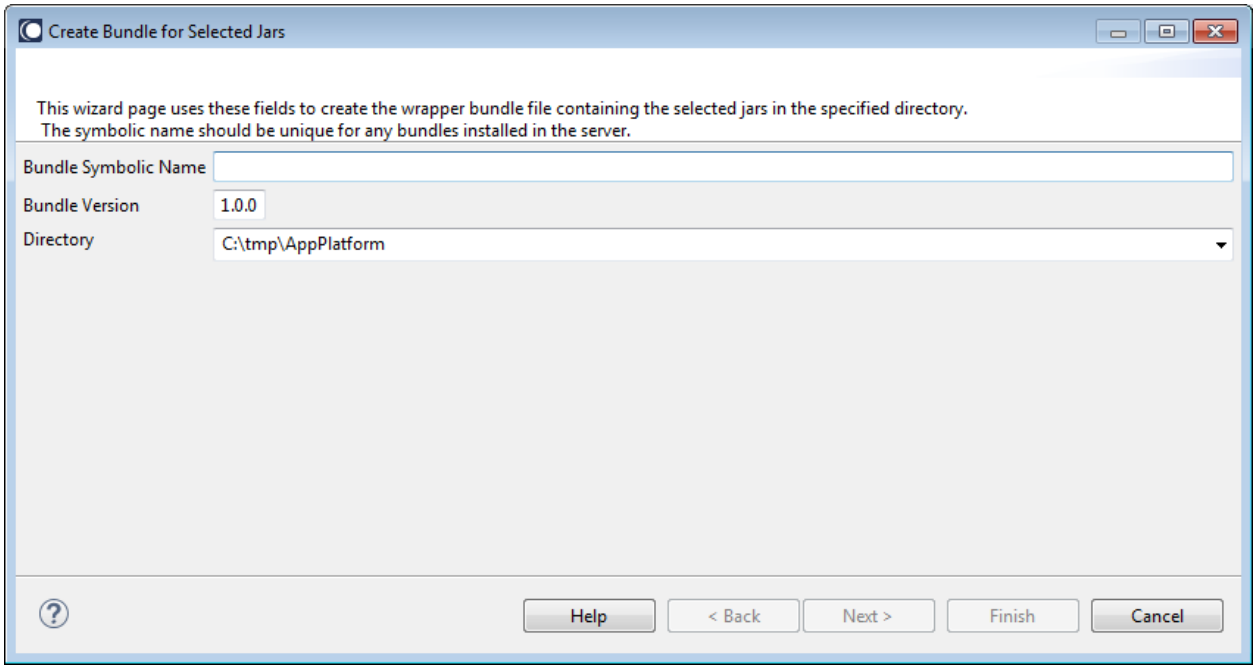
5. Click the OK button to close the Preferences dialog, apply your changes and return to the Bundle Manager view.
6. By default, the Bundle Manager view displays all the local plain jars and bundles in the directories you added to the Bundle Manager Settings page as shown in the example in figure 7-4 below.

Figure 7-4: The Bundle Manager view showing all local jars and bundles available in all the directories defined in the Bundle Manager Setting's page - they can be distinguished by the different icons.



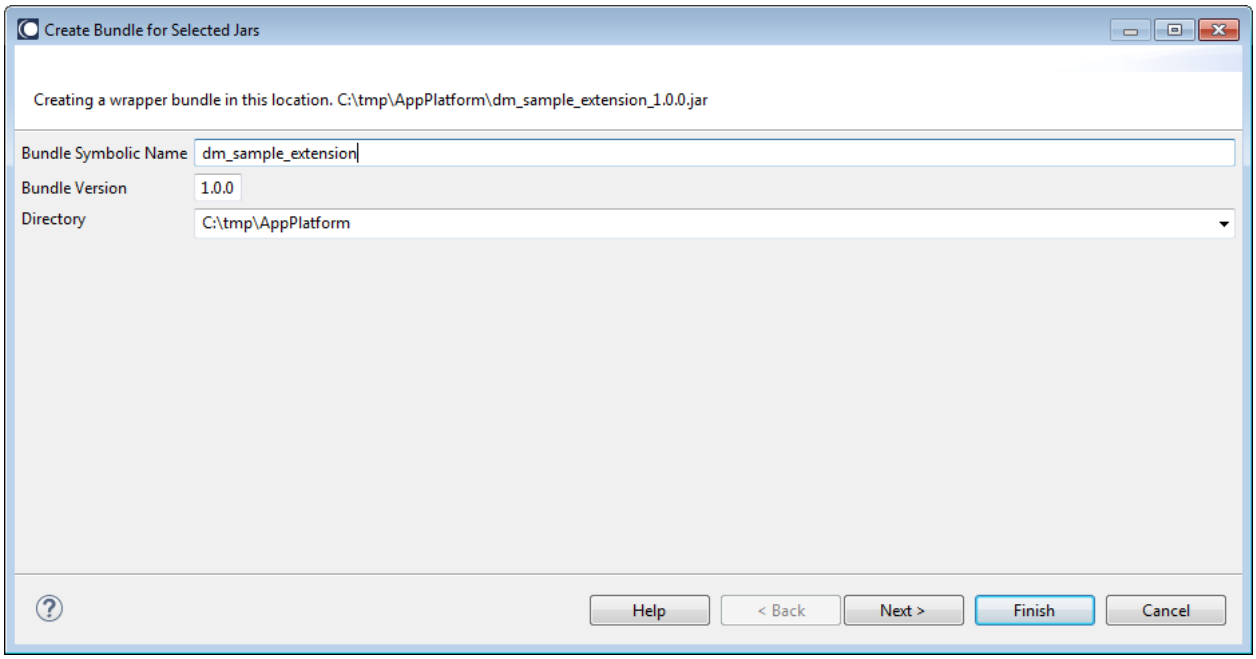
7. Select the dm_sample_extension.jar and click on the plus icon on the Bundle Manager tool-bar. This will open the “Create Bundle for Selected Jars” dialog as shown in figure 7-5 below.

Figure 7-5: OSGi wrapper bundle creation dialog properties configuration page



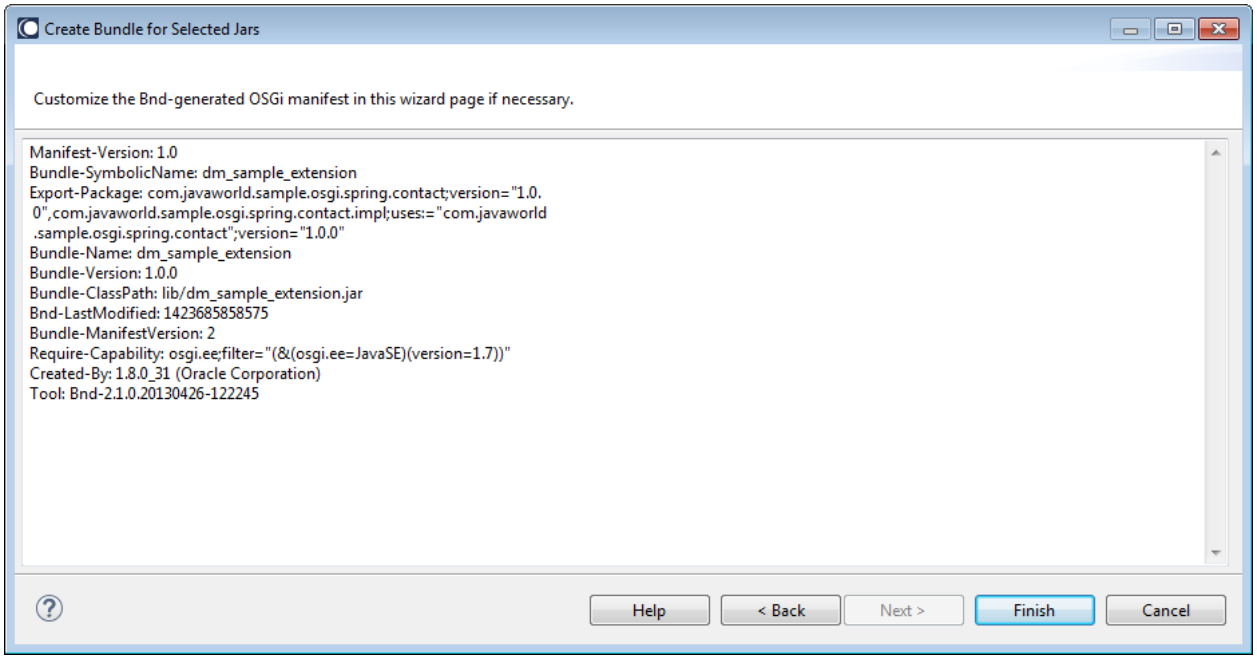
8. Enter a bundle symbolic name, which must be unique, as indicated on the dialog but can be any name you choose. However, it's best to choose a meaningful name (e.g., dm_sample_extension).
    a. The Back, Next, and Finish buttons are disabled until you enter a symbolic name.
    b. After you enter a symbolic name the text message at the top of the dialog above the symbolic name changes to that shown in figure 7-6.

Figure 7-6: Shows that the information message at the top of the dialog has changed once the symbolic name is entered.



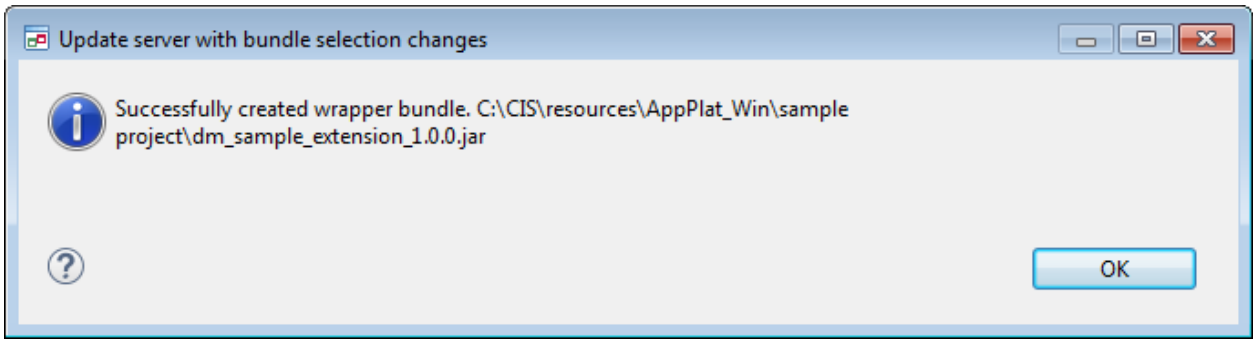
9. Leave the bundle version as the default 1.0.0. This should initially be okay but may need to be updated if you already have an existing bundle with the same version deployed in the server.
10. Choose a directory where you want the wrapper bundle to be created. The list of directories to choose from comes from those you have defined in the Bundle Manager Settings page.
11. To go to the next page, click the "Next >" button after you have defined all your properties, and see the generated manifest file. This takes a few seconds to process. You can customize the generated manifest if needed.
12. Quickly scan the manifest to confirm that it is correct. In particular, check for required headers and any exported packages that you know are required. This page looks as figure 7-7 below.

Figure 7-7: Showing the manifest page in the wizard



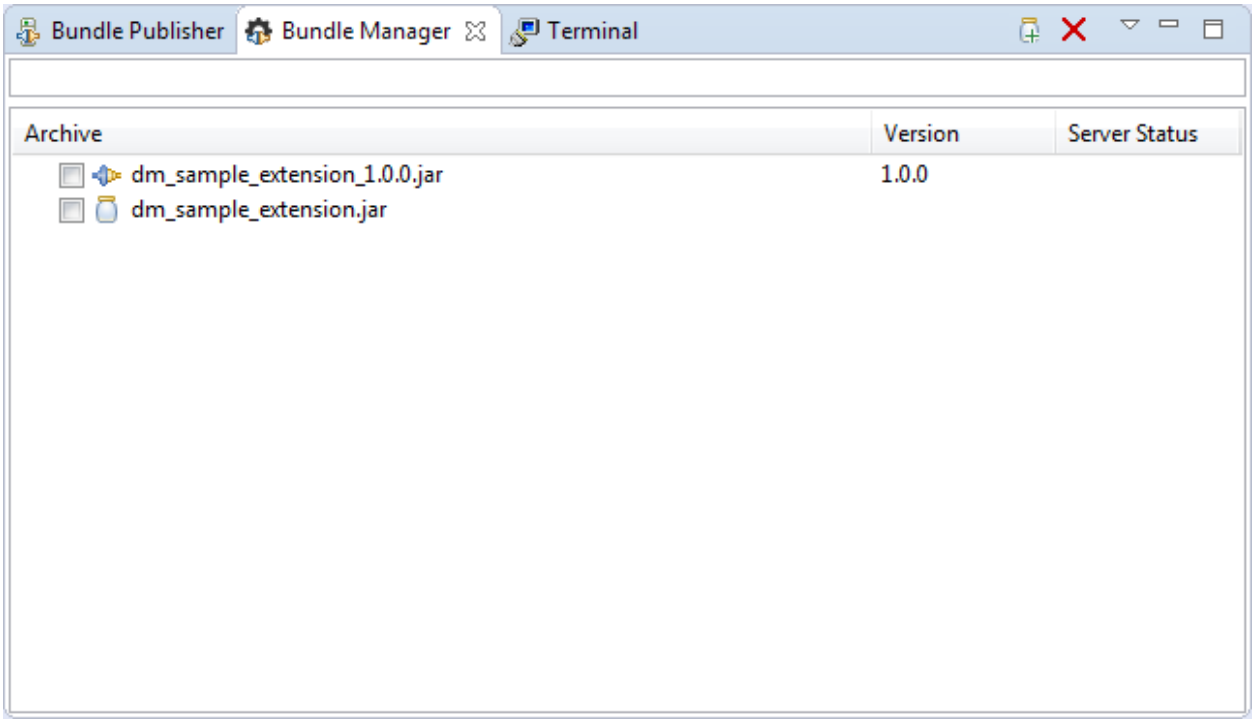
13. Click the Finish button to create the bundle. The “Update server with bundle selection changes” confirmation dialog shown in figure 7-8 below is displayed when you click the Finish button and if all goes well.

Figure 7-8: “Update server with bundle selection changes” confirmation dialog displayed after successful creation of a wrapper bundle.



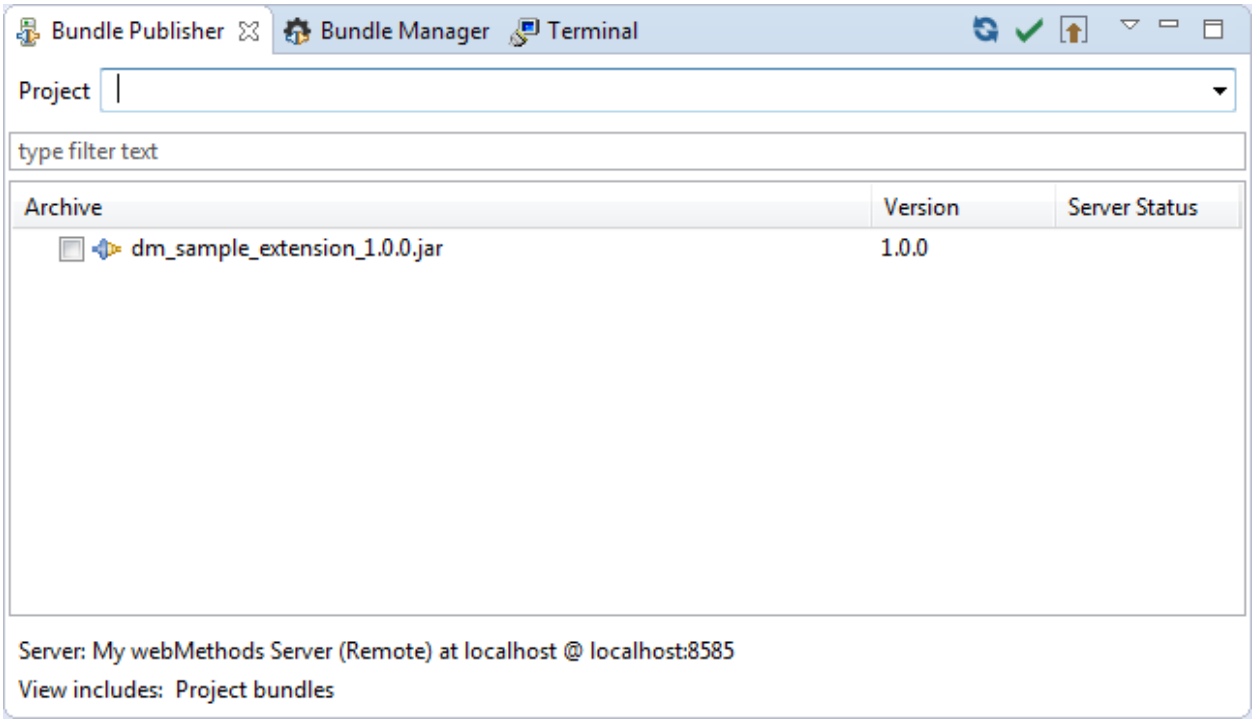
14. Click the OK button to close the “Update server with bundle selection changes” confirmation dialog and return to the Bundle Manager view. You will now see your dm_sample_extension_1.0.0.jar bundle included in the list of jars in the Bundle Manager view as shown in figure 7-9 below.

Figure 7-9: The Bundle Manager view showing the dm_sample_extension_1.0.0.jar bundle included in the list.



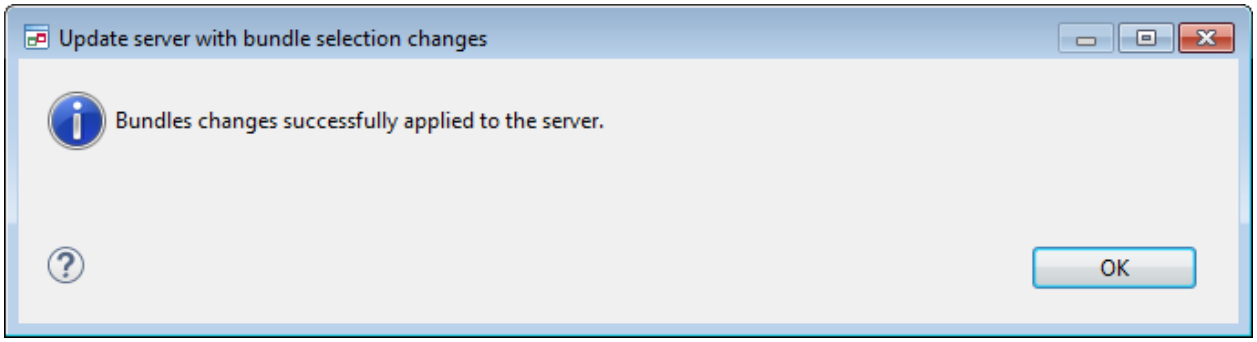
15. Navigate to the Bundle Publisher view to see your dm_sample_extension_1.0.0.jar bundle included as one of the bundles that can be published to the server as shown in figure 7-10 below.

Figure 7-10: Bundle Publisher view showing the new dm_sample_extension_1.0.0.jar included as one of the bundles that can be published to the server - there's additional information at the bottom of the view about the server and the bundles displayed



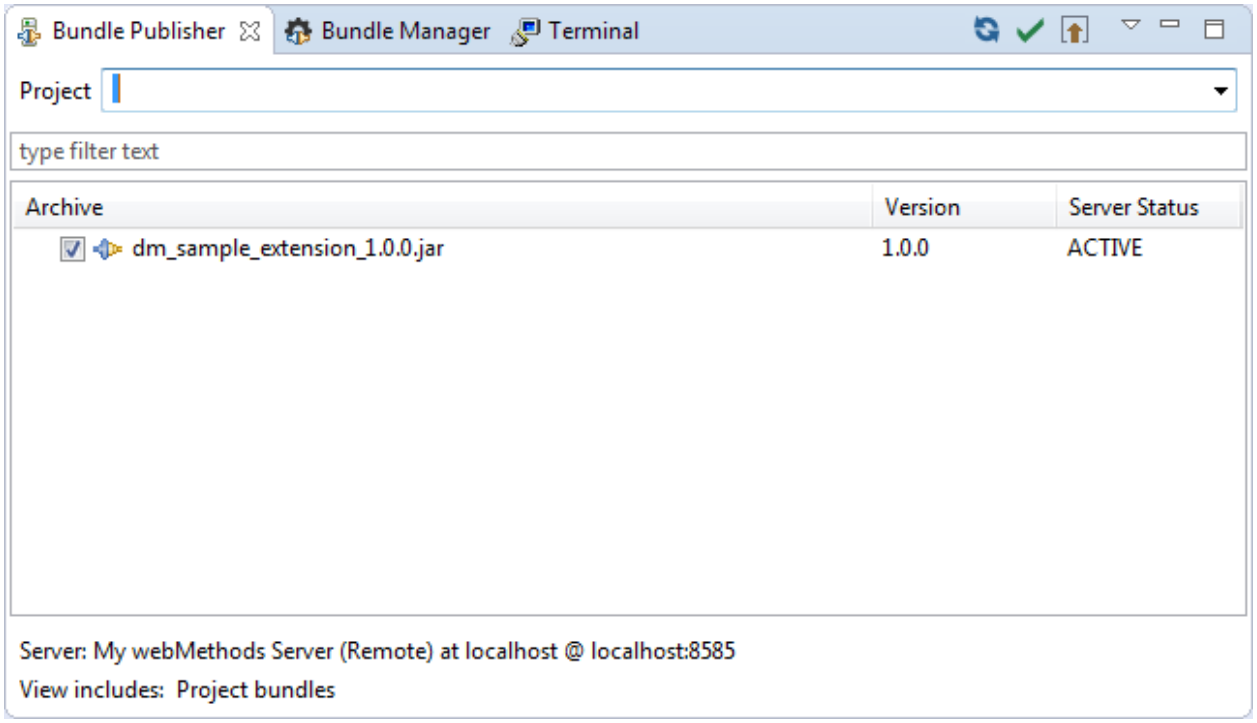
16. The server must be running before you can publish bundles to it. You can start it as defined in chapter 3.8, item 2 above.
17. Once the server has started, select the dm_sample_extension_1.0.0.jar in the Bundle Publisher view and click on the “Update server with bundle selection changes” button on the Bundle Publisher toolbar - the blue save icon third from the left next to the check mark.
18. The confirmation dialog in figure 7-11 below is displayed to show that the bundle was published successfully. Click OK to close it.

Figure 7-11: Confirmation dialog displayed when bundle publishing is successful



19. The dm_sample_extension_1.0.0.jar bundle server status should now be ACTIVE as shown in figure 7-12 below.

Figure 7-12: Bundle Publisher view showing the dm_sample_extension_1.0.0.jar bundle status as ACTIVE.



7.7 Using Application Platform Shared Bundles Tool

Before you can compile the sample1 project, you need to include the dm_sample_extension_1.0.0.jar bundle in the classpath. We don't want to include the dm_sample_extension_1.0.0.jar bundle inside the project, since it is already deployed to the server. To add it to the classpath, use the Application Platform Shared Bundles tool as follows:

7. Right-click on the sample1 project and select Properties to open the "Properties for sample1" dialog.
8. Select "Java Build Path" and then select the Libraries tab in the "Properties for sample1" dialog.
9. Click on the "Add Library…" button to open the standard "Add Library" dialog.
10. Choose "Application Platform Shared Bundles" and click the Next > button to go to the Application Platform shared bundles configuration page.
11. An Edit Variable Entry dialog is displayed with a predefined BUILD_EXTERNAL_DIR variable, as shown in figure 7-13 below. Here the user can select or type the path to the dm_sample_extension_1.0.0.jar third-party wrapper bundle.
    a. This will only happen the first time. Subsequently that path will be automatically filled in and the user can update it.

Figure 7-13: The Application Platform shared bundles configuration page with the Edit Variable Entry dialog displayed.

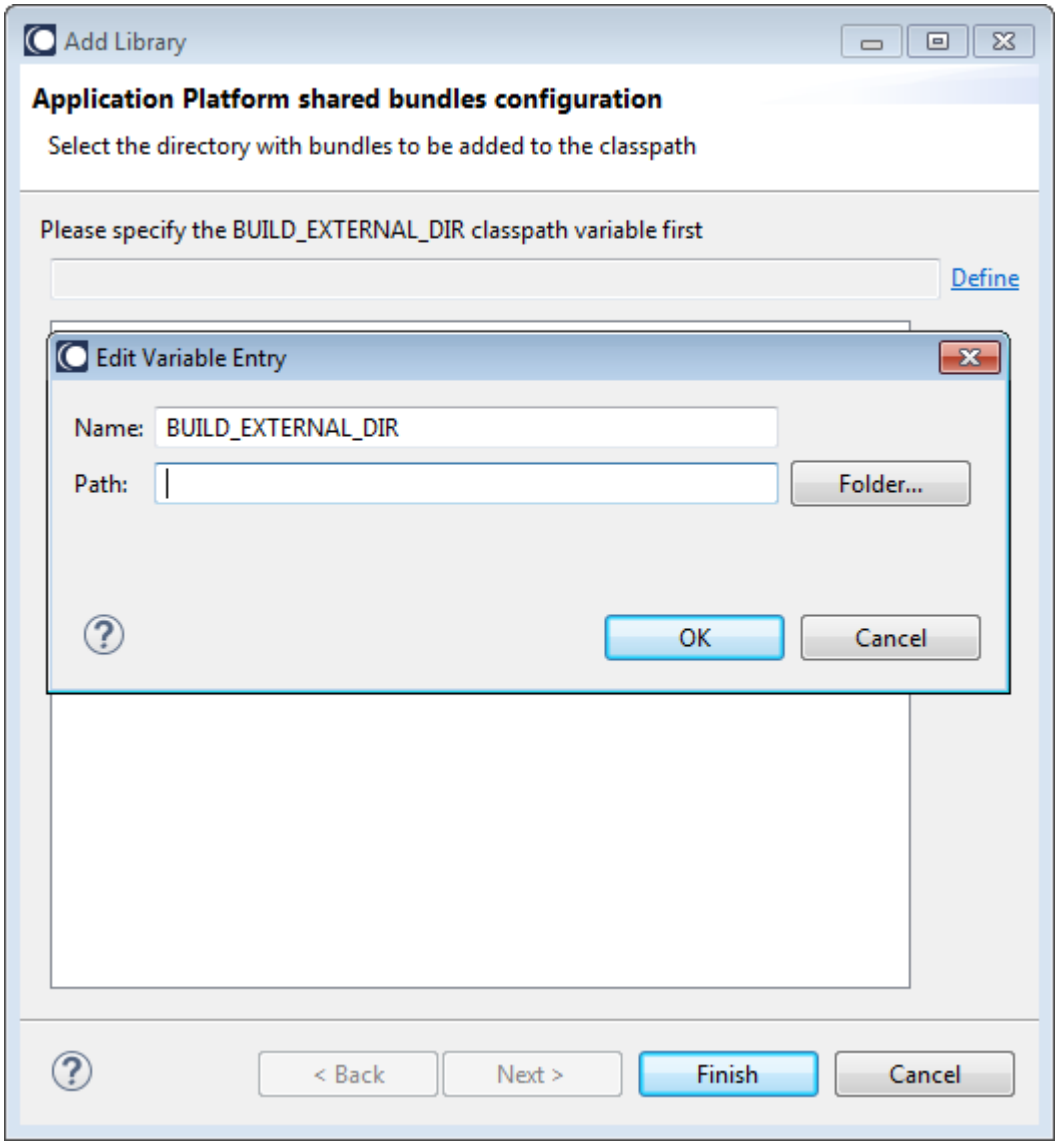
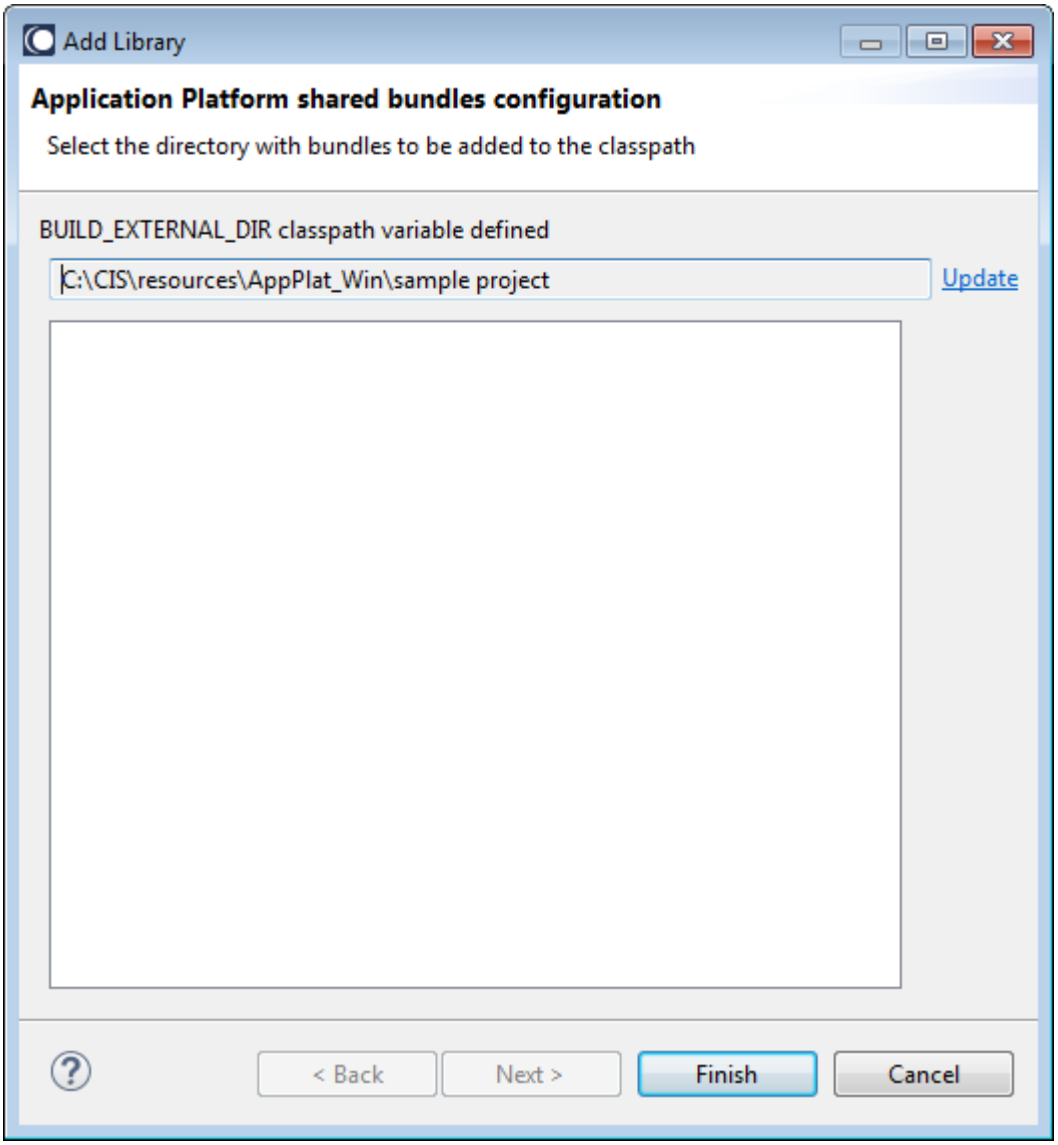
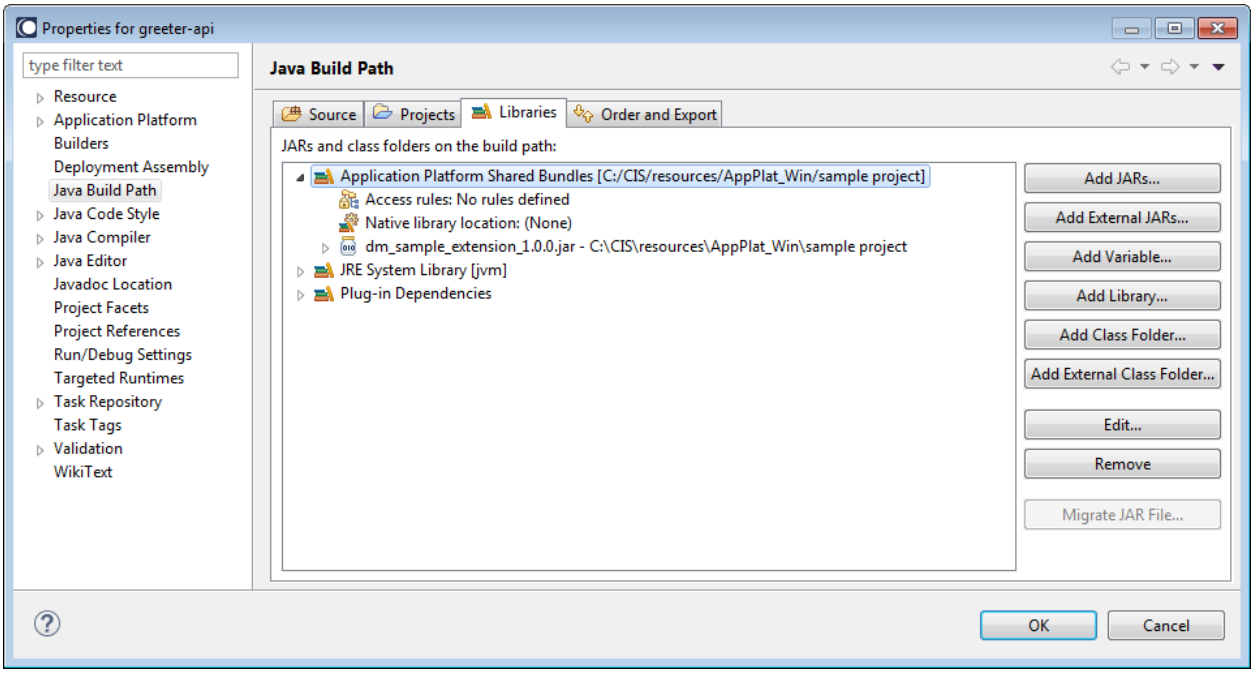


Figure 7-14: The Application Platform shared bundles configuration page after the BUILD_EXTERNAL_DIR path has been selected



12. Click the Finish button to close the dialog and add the Application Platform Shared Bundles folder to the classpath as shown in figure 7-15 below. It shows the Application Platform Shared Bundles folder (C:\CIS\resources\AppPlat_Win\sample project) added to the build path and expanded to show the included dm_sample_extension_1.0.0.jar bundle.

Figure 7-15: “Properties for sample1” dialog showing Application Platform Shared Bundles folder (C:\CIS\resources\AppPlat_Win\sample project) added to the build path and expanded to show the included dm_sample_extension_1.0.0.jar bundle.



13. Click the OK button to close the “Properties for sample1” dialog and return to the Eclipse workspace.

This should solve the compilation errors that you were seeing earlier once you compile the project.

7.8 Compiling the Project

Before compiling the sample1 project you need to generate a manifest file.

1. Right-click on sample1 project and select App Platform > Create Project Manifest to create a manifest file. This file is placed under the src/main/resources/META-INF folder.
2. Double click the MANIFEST.MF file to open it.
3. Click on the MANIFEST.MF tab at the bottom of the open manifest editor.
4. Confirm that it has the following text in the Import-Package header.

```
Import-Package:
com.javaworld.sample.osgi.spring.contact;resolution:=optional;version="[1.0,2)",com.javaworld.sample.osgi.spring.contact.impl;resolution:=optional;version="[1.0,2)"
```

5. Save the project and compile as in chapter 3.7 above. The only difference being to select the sample1 check box if you want to only build this project.

7.9 Publishing the Project to Integration Server

This section is the same as in chapter 3.8 above. The only difference is to choose the sample1 for publishing.

7.10 Accessing the Project on the Web

Access the sample1 application using a web browser.

1. Open any browser that you have installed on your local machine.
2. Type http://localhost:8072/sample1/ShowName.jsp to open your web client.
3. Start using the application.
    a. Enter a first name and last name and click the Get button to see the results - you should see the combined first and last name printed below the “Contact’s Full Name” label. Any new name you enter is published below this label.

7.11 Unpublishing the Project

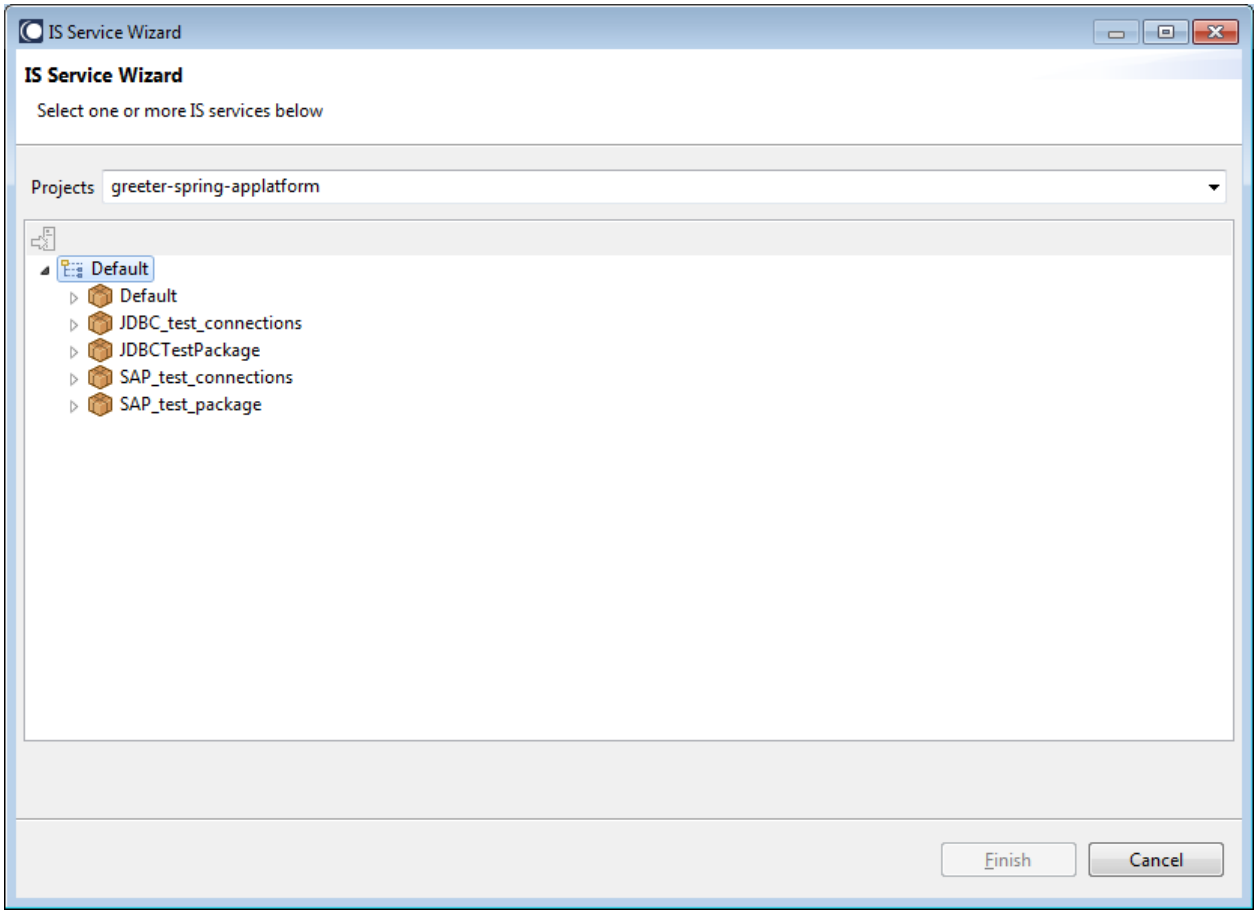
This section is the same as in chapter 3.10 above. The only difference is to choose sample1 for un-publishing.

8 Generating Java Objects from IS Services in Application Platform

Plain old Java objects (POJOs), to be used in developing business applications within the Application Platform perspective, can be generated from IS services as follows:

1. First start the server by right-clicking on “webMethods Integration Server at localhost [Stopped]” and clicking Start on the popup menu.
    a. The focus may switch to the Console view until the server has started and then switch back to the Servers view. The focus will switch to the Console view if there are errors.
    b. The Servers view should now display “webMethods Integration Server at localhost [Started, Synchronized].”
2. Right-click on your project in Designer Project Explorer and select Software AG App Platform > IS Tools > IS Service Wizard to open the IS Service Wizard dialog.
    a. You can also use the the CTRL+SHIFT+Z shortcut.
3. Confirm that the project, selected in the Projects drop-down list is the one you want (this should be the default selection). See figure 8-1 below for the example greeter-spring-applatform.

Figure 8-1: IS Service Wizard showing greeter-spring-applatform as the selected project in the drop-down list.



4. Expand the default tree. If Default is still showing as “Default (not connected),” then do the following:
    a. Right-click on Default (not connected) and click on Connect to server to open the Connection Required dialog.
    b. Click the Connect button to connect to the IS server; this will also close the Connection Required dialog.
    c. The Default text should now be display as Default (without the “not connected” text).
5. Select the IS services you wish to convert to POJOs and click the Finish button to create them.
6. The POJOs are placed in the genSource folder of your project.