Building an Agile PLM Web Application with JDeveloper and Agile 93 Web Services

Tutorial

By: Maneesh Agarwal, Venugopalan Sreedharan
Agile PLM Development

October 2009
Chapter 1 Overview

This chapter includes the following:

- About this Tutorial ................................................................. 4
- Time to Complete ................................................................. 4
About this Tutorial

In this tutorial, you will use Oracle JDeveloper 11.1.1.1.0 and ADF (Application Development Framework) components to build a customized Web application for Agile PLM. You will utilize Web services provided in Agile 93 from JSF pages to extend the supported Agile PLM functionalities. The goal of this tutorial is to demonstrate creating custom user interfaces for Agile 93 using ADF components and Web services.

Time to Complete

The estimated time to complete this tutorial is 45 minutes.
This chapter includes the following:

- Understanding WSDL Files ........................................................................................................................................ 5
- New Topic (14) .................................................................................................................................................... 6
- Preparing the Required Environment .................................................................................................................. 6

Understanding WSDL Files

Files with the WSDL extension contain Web service interfaces expressed in the Web Service Description Language (WSDL). WSDL is a standard XML document type specified by the World Wide Web Consortium. WSDL files are used to communicate interface information between Web service producers and consumers. A WSDL file provides all the information a client application needs in order to use the Web service.

A WSDL description allows a client to utilize a Web service’s capabilities without knowing the implementation details of the Web service.

A WSDL file contains the following information which is necessary for a client to invoke the methods of the Web service:

- The data types used as method parameters or return values
- The individual methods names and signatures (WSDL refers to methods as operations)
- The protocols and message formats allowed for each method
- The URLs used to access the Web service
New Topic (14)

Agile WSDL files describe Web services that are used to obtain Agile metadata, search for business objects, download attachments, and manipulate tables and perform collaboration tasks. In this tutorial you will generate the stubs for Agile Collaboration Web services and design a Web application that will invoke these services. The Web pages designed must be able to authenticate a user, query the status of any routable Agile object, and enable the end user to change the status for the given routable object.

Preparing the Required Environment

Complete the following tasks before starting to build the application:

- Acquire access and a user account to Agile PLM Release 9.3
- Verify the location of the WSDL files for available Web services on the Agile server. You can find a list of these services at http://<Agileserver>:<port>/CoreService/services.
Building the Web Service Application

Complete the following steps to build the Web service application.

1. Open JDeveloper.
2. Select the default role when prompted for your user role at startup.
3. Expand the applications navigator and select New Application.
4. In the Create Application dialog box, enter Agile Web Services for the application name and enter oracle as the application package prefix. Now select Fusion Web Application (ADF) from the Application Template drop-down list.

5. In the Create Application dialog box, type Agile Web Services for the application name and oracle for the application package prefix.

6. Select Fusion Web Application (ADF) from the Application Template drop-down list.

7. Click Finish to complete creating of the application.
The Application Navigator shown below will appear.

![Application Navigator Image]

**Building the Web Service Proxy**

Use the following procedure to create a Web service proxy for Agile Collaboration services.

1. **Right click on Model** and then select **New**.
The New Gallery wizard opens.

2. In New Gallery wizard, do as follows:
   a. Select All Technologies tab.
   c. In Items pane, select Web Service Proxy.
   d. Click OK to proceed

   The Create Web Service Proxy wizard opens.

3. In Create Web Service Proxy wizard do as follows:
   a. Click Select Client Style and select JAX-RPC Web Logic Style in the right pane.
b. Click **Select Web Service Proxy** and type the applicable URL in **WSDL Document URL**: field for Agile Collaboration services, for example, `http://<AgileServer>:<Port>/CoreService/services/Collaboration?wsdl`, and then click **Next**, and JDeveloper will build the model and generate stubs for this WSDL file.

c. Click **Specify Default Mapping Options** and **Package Name** field: type `collaboration`, and then click **Next** > Run against a service deployed to an external server.

d. Click **Next** twice and then click **Finish** to complete creating the Web Service proxy.
Building an Agile PLM Web Application with JDeveloper and Agile 93 Web Services

The Applications Navigator displays the Web Service proxy files that are generated.

Building the Web Service CollaborationPOJO

Agile Web services use ComplexTypes in both request and response variables. To expose these as user interface components you may create a POJO class where all the necessary fields are declared as variables with accessors. Client code to invoke the web services for getStatus and changeStatus is written as methods in this class. These methods will be exposed as ADF buttons in the web pages.

1. Right click on Model, choose New, and under the category General, choose Java Class.
2. Type **CollaborationPOJO** for class name and then click **OK**.
3. Create two new variables `username` and `password` and then right click and select Generate accessors.

4. Declare two variables to hold the Agile class identifier and Agile object number values and call them `classidentifier` and `objectidentifier`. Similarly, declare a variable named `newstatus` to use while invoking the `changeStatus` Web service.

   **Note** These are necessary to get the status of the routable object with the `getStatus` Web service.

5. Declare two variables for the Web service responses; one of type `GetStatusResponseType` and another of type `ChangeStatusResponseType`.

6. Generate accessors for all declared variables.

7. Write the methods `getStatus` and `changeStatus` which consisting of client code using the Web service proxy to invoke the Web services.

   Username and password information must be set on the Web service port. Requests are formed using the necessary input values and web services are invoked on the Web service proxy `porttype` object. For more information on writing Agile Web service client code, refer to Agile 93 Web service manual.
You can refer to the following sample code to test the Web service client code.

```java
private String classIdentifier;
private String objectIdentifier;

private String newStatusIdentifier;
private String commentString;

private GetStatusResponseType getStatusResponse;
private ChangeStatusResponseType changeStatusResponse;

private static CollaborationService_Impl service;
private static Collaboration_PortType myPort;

public void setCredentials() {...}

public void getStatus() {
    try {
        setCredentials();
        GetStatusResponseType request = new GetStatusRequestType();
        AgileGetStatusRequestType[] requestBatch =
            new AgileGetStatusRequestType[1];
        requestBatch[0] = new AgileGetStatusRequestType();

        requestBatch[0].setClassIdentifier(classIdentifier);
        requestBatch[0].setObjectNumber(objectIdentifier);
        request.setStatusRequest(requestBatch);

        System.out.println("invoking ws for " +
                          requestBatch[0].getClassIdentifier() + ";" +
                          requestBatch[0].getObjectNumber());
        getStatusResponse = myPort.getStatus(request);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

For information about complex requests and responses in ADF, refer to article and its references at this site:

Also, JDeveloper 11g provides support for complexTypes, the following article describes consumption of Agile web services using the inbuilt support for complex input parameters. Refer to article and its references at this site:

In this tutorial however, we will restrict our discussion to the usage of POJO classes for handling complexTypes.

Creating a data control for the POJO

To create a data control for the POJO façade, complete the following steps:

1. Right click on the POJO class CollaborationPOJO.
2. Click on Create Data Control.

   The data control for CollaborationPOJO listed in the Data Controls pane appears.

Designing the Web Page

To design the Web pages of the application, use a JSF page flow and configuration file as shown below.

1. Right click on the ViewController project and click New.
2. Under the Web Tier category, select JSF and choose JSF Page Flow and Configuration.
This will create the file `faces-config.xml` to configure the Web pages.

3. Select and drag the JSF Page component from the components palette to the `faces-config` diagram; rename the JSF page and add two more pages and rename them as shown below.

Creating Navigation Cases

This step creates a navigation case from the first JSF page to the second. To do this, do as follows:

1. Drag the JSF Navigation Case component onto the first JSF page and draw a line onto the second.

Rename the case from the default label `success` to `getstatus`.
2. Similarly, add three more navigation cases as shown below:

Designing the Index Page

1. Double click on JSF pages in the faces-config diagram view.
2. Double click on `index.jsf`.
3. Choose Oracle Three Column Layout for the initial page layout.
4. Expand the Page implementation node and make sure Do not automatically expose UI components in a managed bean is selected.
5. Expand the CollaborationPOJO node in the DataControls panel and drag the username field to the left pane of the page.
6. Select Create > Texts > ADF Input Text w / Label.

7. Drag the password field in the left pane and drag classidentifier and objectidentifier onto the middle pane.

8. Change the password input text’s appearance to secret using the property inspector.

9. Drag the operation getStatus onto the middle pane and select Create > Methods > ADF (button).

   This operation is bound to the method in the POJO class for invoking the getStatus Web service that queries the status of any Agile PLM routable object.

10. In property inspector, for the button, edit the Action value and choose 'getstatus' which is the navigation case you defined in faces-config.xml.
When you are done, your page should now look like this.

Designing the StatusInfo Page

This is similar to the previous step. Go back to faces-config.xml and double click on StatusInfo.jspx. Define this page as you did the index page. Now we are ready to display the results from the Web service operation.

1. Expand the node getStatusResponse in the Data Controls pane.
2. Expand StatusCode and drag the value element to the right pane of the Web page. This value indicates whether the Web service call executed successfully or not.
3. In getStatusResponse, select and drag statusResponse to the middle panel and choose Create-Tables->ADF Read Only Table.
4. Accept the default values as shown below and click **OK**.

5. Similarly, drop the element exception under **Exceptions** node and the element warning under **Warnings** node as read only tables. These will provide warnings or exceptions that are encountered while invoking the Web service.

6. Drag a button from the component palette and label it **Back**.

7. In property inspector, edit the Action property and in Navigation case, select **back** defined earlier.
Testing the Web Application

This chapter includes the following:

- Procedures

Procedures

The test involves checking the status of an ECO called 'C00001. To test the Web application, do as follow:

1. Click on Save All.
2. Open index.jspx, right click and choose Run.
3. This starts the Web Logic server, if it is not already running and will deploy your application before automatically bringing up the Web page for testing. Our index page now looks like this:

4. Type your Agile username and password and an ECO called C00001.
5. Click getStatus to query for the status of ECO C00001.
The status information page displaying the information that was retrieved from the result of the Web service query appears.

6. Expose the variable `newStatusIdentifier` from `CollaborationPOJO DataControls` as an input on the `StatusInfo` JSF page.

7. Drag the operation `changeStatus` as an ADF button and in action property choose the navigation case `changestatus`.

   The button will invoke the change status Web service and redirect the user to the third JSF
page ChangedStatus.jsfx.

8. After defining ChangedStatus.jsfx, drag the status code element as an output text component to the web page. Using this value the end user can determine whether the object was successfully routed to a new state.

After these changes the web page would look like this:
If the change status Web service succeeds, then the following page appears.

| Status Code: SUCCESS |
Chapter 5

Concluding Remarks

This chapter includes the following:

- Tutorial Objectives
- Other Sources of Information

In this tutorial you created a Web application to query and change the status of Agile routable objects using the capabilities provided by Agile 93 Web Services. In this process, you learned to:

- Build the data model for Agile web services from a WSDL file
- Expose the necessary Web services through a POJO facade for which you created the data controls
- Create Page Flows and navigation cases for Index, Status Info, and Change Status Web pages
- Run the JSF pages and utilize Agile Web Services

Tutorial Objectives

In this tutorial you created a Web application to query and change the status of Agile routable objects using the capabilities provided by Agile 93 Web Services. In this process, you learned to:

- Build the data model for Agile web services from a WSDL file
- Expose the necessary Web services through a POJO facade for which you created the data controls
- Create Page Flows and navigation cases for Index, Status Info, and Change Status Web pages
- Run the JSF pages and utilize Agile Web Services

Other Sources of Information

To learn more about Oracle ADF and JDeveloper, can refer to: