ORACLE

Reference: 2006/01/24-10.1.2.0.2

Oracle BPEL Process Manager 10.1.2.0.x

Quick Start Tutorial

This document describes how BPEL and the Oracle BPEL Process Manager facilitate development of SOA applications through composing synchronous and asynchronous services into end-to-end, standard BPEL process flows. It is not intended to be a complete development guide, but rather a tutorial and guided tour providing a rapid overview of many of the most commonly used features. Pointers are given throughout to additional documentation and samples and readers should also use the many other documents available at http://otn.oracle.com/bpel after going through this guided tour.

Note: Much of this document is a tutorial with instructions for you to work hands-on with a local installation of the Oracle BPEL Process Manager. However, you can also just use this document to get some of the "test drive experience" even if you do not plan to install our server at this time.

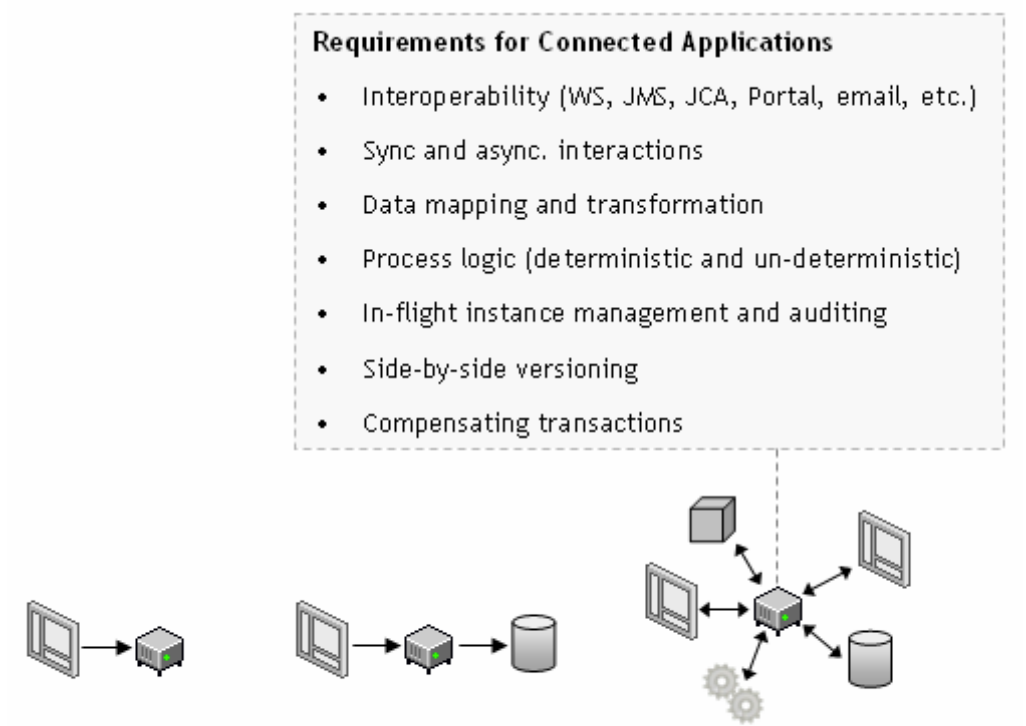
This document has been tested with both the 10.1.2.0.0 and 10.1.2.0.2 releases of Oracle BPEL Process Manager.

Contents

BPEL, the Cornerstone of SOA

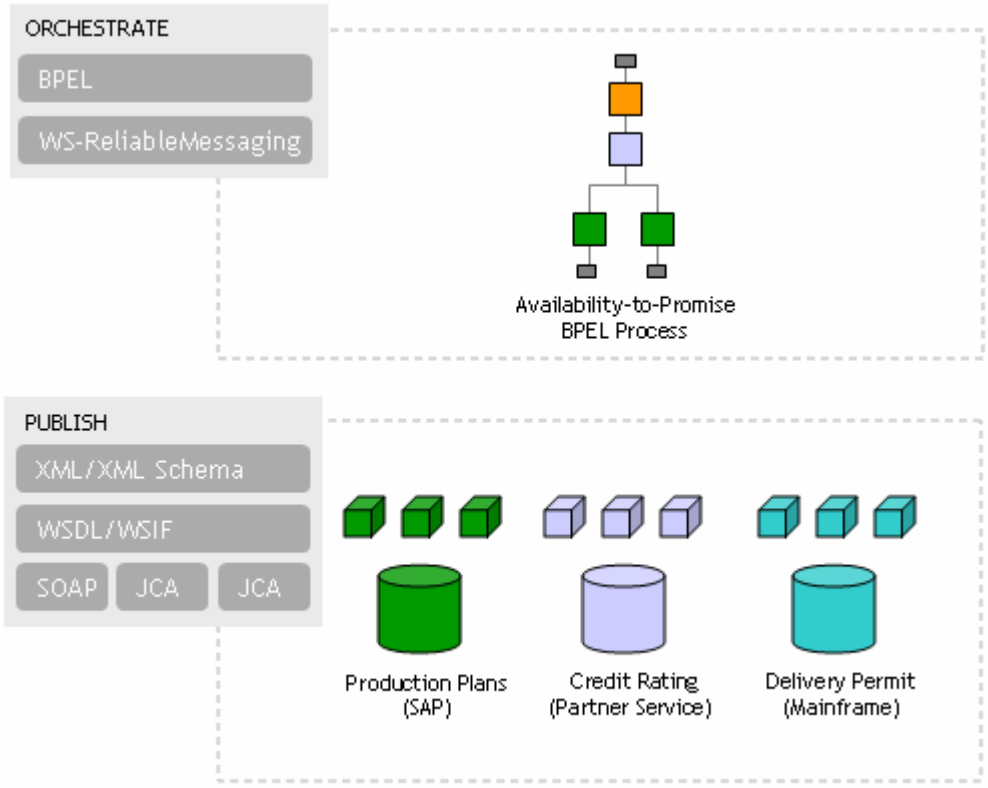
TOWARDS CONNECTED APPLICATIONS

An increasing number of companies are looking at SOA and Web services as an architectural blue-print and a set of standards for addressing the integration requirements involved in building connected applications. While SOA has been a best practice for over a decade, there has been confusion around which standards to adopt. BPEL and Web services standards have solved this problem by addressing common application requirements in an open, portable and standard way. SOA enables business agility by maximizing leverage of existing resources while minimizing the cost of deploying new services into a business process. Enterprises adopting these standards and architectural approach are already achieving significant ROI from using the same standards-based approach to building connected applications that they have used for building web applications with Java/J2EE.



Making web services work is a 2-step process. First you publish your services and then you compose, or orchestrate, them into business flows. Publishing a service means taking a function within an existing application or system and making it available in a standard way, while orchestration is composing multiple services into an end-to-end business process. The Web services standards, including WSDL, XML and SOAP, have emerged as an effective and highly interoperable platform for publishing services. In addition, high

performance binding frameworks allow enterprises to access legacy systems and native Java code without necessarily having to wrap them in a SOAP interface.



KEY BPEL CONCEPTS

BPEL (Business Process Execution Language) has emerged as the clear standard for composing multiple synchronous and asynchronous services into collaborative and transactional process flows. BPEL benefits from 15+ years of research poured into its predecessor languages, XLANG and WSFL.

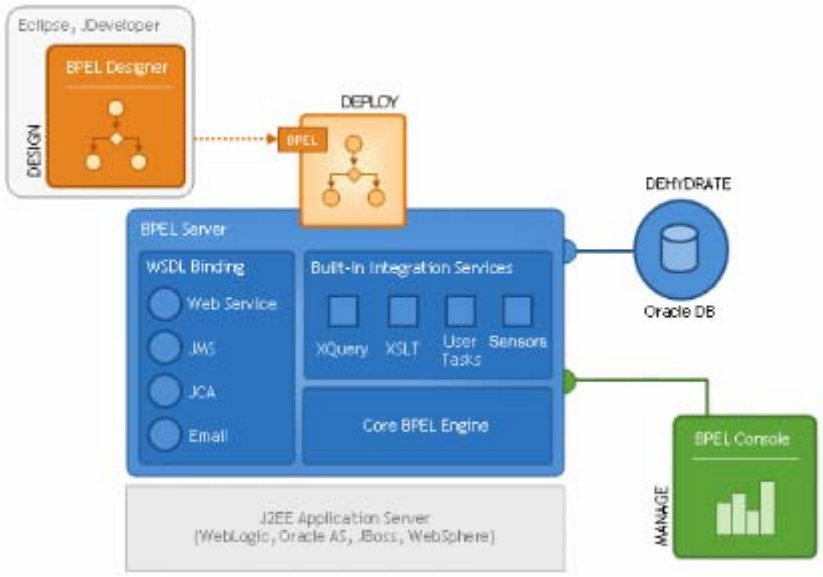
- Web Services/WSDL as component model
- XML as data model (data loose-coupling)
- Sync. and async. message exchange patterns
- Deterministic and non-deterministic flow coordination
- Hierarchical exception management
- Long-running unit of work/compensation

Since the BPEL specification was submitted to OASIS in March 2003, it has gained the support of nearly every major industry vendor (including Oracle, Microsoft, IBM, SAP, Siebel, BEA and Sun). With the list of vendors committing to support for BPEL growing every day and several BPEL server implementations publicly available, the standards war in the business process space appears to be over. And this is a great thing for enterprises who can now implement their business processes in a standard and portable way, avoiding vendor lock-in to a degree that has not been possible before.

The Oracle BPEL Process Manager

INTRODUCTION

The Oracle BPEL Process Manager provides a developer-friendly and reliable solution for designing, deploying and managing BPEL business processes.



The **BPEL Designer** provides a graphical and user-friendly way to build BPEL processes. What is unique about the Oracle BPEL Designer is that it uses BPEL as its native format. This means that processes built with the Designer are portable and in addition it enables developers to view and modify the BPEL source at any time.

The core **BPEL engine** provides the most mature, scalable and robust implementation of a BPEL server available today. The Oracle BPEL Process Manager executes standard BPEL processes and provides a "dehydration" capability so that the state of long-running flows is automatically maintained in a database, enabling clustering for both fail-over and scalability. The BPEL Server leverages an underlying J2EE application server, with support for most major commercial application servers and a bundled version available.

The **built-in integration services** enable developers to easily leverage advanced connectivity and transformation capabilities from standard BPEL processes. These capabilities include support for XSLT and XQuery transformation as well as bindings to hundreds of legacy systems through JCA adapters and native protocols. Human Workflow is provided as a built-in BPEL service to enable the integration of people and manual tasks into BPEL flows. The JDeveloper BPEL Designer provides wizards to build these complex workflows and a simple GUI to map transformations.

The extensible **WSDL binding framework** (DOS WSIF – http://www.apache.org/) enables connectivity to protocols and message formats other than SOAP. Bindings are available for Java, JMS, email, JCA, HTTP GET and POST and many other protocols enabling simple connectivity to hundreds of back-end systems. WSIF allows you bind directly and transparently to any backend protocol or programming construct, so you get the benefits of a loosely-coupled Web services architecture with the performance and transactionality of native protocols.

The **BPEL Console** provides a mature web-based interface for management, administration and debugging of processes deployed to the BPEL server. Audit trails and process history/reporting information is automatically maintained and available both through the BPEL Console and via a Java API.

HOW IS IT DIFFERENT?

The power of an open standard - By capturing your business processes in BPEL, you protect your intellectual property and investments while avoiding vendor lock-in. BPEL is to business process management what SQL was to data management.

Unparalleled visibility and administration - The BPEL Console reduces the cost and complexity of deploying and managing your business processes: Visually monitor the execution of each BPEL process, drill down into the audit and view the details of each conversation or debug a running flow against its BPEL implementation

Dramatic cost savings - The Oracle BPEL Process Manager is 60-80% less expensive to buy and maintain than traditional EAI solutions.

Open and flexible binding framework - Orchestrate XML Web services, but also Java/J2EE components, portals, JCA interfaces and JMS destinations. Tie into 100+ back end systems. Leverage your Java skills and application server investments.

Production Ready - The BPEL Server leverages an underlying J2EE application server so that using BPEL as the integration and orchestration layer, seamless enterprise-wide application solutions can be built up, reusing and extending existing heterogeneous systems. Customers have been in production since 2003 with most major verticals represented. However, early adoption has been particularly robust in telecommunications, financial services, health care, high tech manufacturing, government, defense and aerospace.

FEATURE SUMMARY

BPEL DESIGNER

- Native BPEL Support
- Drag-and-drop process modeler
- UDDI and WSIL service browser
- Visual XPATH editor
- One-click build and deploy

BUILT-IN INTEGRATION SERVICES

- Java embedding
- Email and JMS messaging services
- XSLT and XQuery transformation services
- User task manager and portal integration
- Extensible WSIF binding framework

BPEL CONSOLE

- Visual Monitoring
- Auditing
- BPEL Debugging
- In-flight Administration
- Performance Tuninng
- Partitioning/Domains

BPEL SERVER

- Comprehensive BPEL 1.1
- Sync. And async messaging
- Context Dehydration
- Advanced Exception Management
- Side-by-side versioning
- Large XML documents

Installation Instructions

Theses instructions are based on the10.1.2 release of the Oracle BPEL Process Manager, and will guide you through the developer edition installation.

PREREQUISITES

- Windows XP, Windows 2000 with Service Pack 3, Windows 2003 with Service Pack 1, Sun SPARC Solaris version 8 and 9, Red Hat Enterprise Linux AS/ES 2.1 and 3.0, or SUSE Linux Enterprise Server 8 and 9.
- 1 GB of disk space, a minimum 1 GB RAM and minimum 1535 MB of swap space.
- Internet Explorer 6.0
- Monitor configured to display at least 256 colors.

DOWNLOAD AND INSTALL

The Oracle BPEL Process Manager is available for download at http://otn.oracle.com/bpel.

This Quick-Start guide uses the **Oracle BPEL Process Manager for Developers** installation type, which installs both the JDeveloper BPEL Designer and the Oracle BPEL Process Manager. The installation will also include and deploy a stand-alone Oracle Containers for Java (OC4J) instance and an Oracle Lite Database.

The same installer can be used to deploy a Middle-Tier installation of Oracle BPEL Process Manager. If you wish to use this install type you would previously need to have installed Oracle Application Server (J2EE and Wireless as a minimum) and an Oracle Database configured with the Oracle BPEL Process Manager schema.

Most of the instructions here apply equally to the WebLogic and JBoss (and WebSphere when available) versions of Oracle BPEL PM. For additional details, or for users installing on platforms other than Windows, see the appropriate *Oracle Application Server Integration* Installation Guide.

Install Oracle BPEL Process Manager for Developers:

1 To begin the installation process, log onto the host where you wish to install the Oracle BPEL Process manager.

2 Start the Oracle Universal Installer from the *bpel* directory of the CD-ROM or staging directory by clicking *setup.exe*.

The Welcome screen appears.

3 Click **Next**.

The Specify File Location screen appears.

If your host is detected to be part of a cluster, the Specify Hardware Cluster Installation Mode screen appears. Select **Noncluster Installation**. This installs Oracle BPEL Process Manager on this node only.

4 Accept the default Name and Path, or specify a new Oracle home name and directory path in which to install Oracle BPEL Process Manager components.

Do not use an existing home or directory path.

Do not change the directory path in the **Source** field. This is the location of the installation files.

5 Click **Next**.

The Select Installation Type screen appears.

6 Select **BPEL Process Manager for Developers** and click **Next**.

The Specify Outgoing Proxy Information screen appears.

7 If you have a direct connection to the Internet and do not use a proxy server, or if you accept the default information, then click **next**.

Otherwise enter details pertaining to your proxy server. In the **Bypass Proxy for Addresses** field leave the tag **<local>** to ensure that your hostname is automatically included in the list.

8 The Summary screen appears, click **Next**.

When the installation is complete the End of Installation screen appears with information for your review.

9 Click **Exit** and confirm when prompted.

10 See the following URL for information on further patches as may be needed.

http://otn.oracle.com/bpel

Now you will be able to run your Oracle BPEL Process Manager and BPEL Designer, as shown below.

To start the BPEL Designer:

1 Start the BPEL Designer from the Windows Start menu by clicking **Start > Programs > Oracle – *Oracle_Home* > BPEL Process Manager 10.1.2 > JDeveloper BPEL Designer**.

To start the BPEL Server:

You should also start your Oracle BPEL Server, since you will be using it to deploy and test flows that you create with the Designer.

2 Start the Oracle BPEL Process Manager from the Windows Start menu by clicking **Start > Programs > Oracle – *Oracle_Home* > BPEL Process Manager 10.1.2 > Start BPEL PM Server**.

Example I: Your First BPEL Process

In this section of the guided tour, you will learn how to use the Oracle BPEL Designer to build, deploy, and test your first BPEL process. The process is an asynchronous flow that calls a simple service: a synchronous credit rating service. Creating this process is intended to be the first step toward building a more sophisticated application (like the loan flow example).

GETTING STARTED

Before starting the main content of this tutorial, you should compile and deploy the credit rating service that you will invoke as part of the tutorial, making it available on your local BPEL Server. You will use the command line rather than the BPEL Designer to compile and deploy this service. In general, all BPEL processes can be compiled either through the BPEL Designer or command-line Ant interfaces.

To compile and deploy the credit rating service:

1 Open up a developer command prompt if you do not already have one open.

Start > Programs > Oracle – *Oracle_Home* > BPEL Process Manager 10.1.2 > Developer Prompt.

2 Navigate to the Credit Rating Service Directory

```
> cd utils\CreditRatingService
```

3 Execute the **obant** command.

```
> obant
```

To test the credit rating service:

4 If you have not already done so, connect to the BPEL Console (at http://localhost:9700/BPELConsole) and log in.

An alternative is to use the Windows Start menu and click

Start > Programs > Oracle – *Oracle_Home* > BPEL Process Manager 10.1.2 > BPEL Console.

5 In the **Name** column (under the **Dashboard** tab of the console), click the link for the `CreditRatingService` BPEL process.

6 In the test form that appears, enter any nine-digit number as the social security number (for example, 123456789) and click **Post XML Message**.

You should get back an integer credit rating (or a fault if the social security number you entered began with a 0). In either case, the service is confirmed to be installed successfully.

To set JDeveloper proxy information:

First you should set the hostname of your computer in your JDeveloper preference settings. If you do not do this, you can receive parsing errors when selecting a WSDL file on the WSDL Chooser window while creating a partner link.

7 Select Preferences from the JDeveloper Tools main menu

8 Click Web Browser and Proxy

9 Make sure that your hostname and localhost (at least) are in the Exceptions field. For example if your hostname is myhost-pc:

us.acme.com|*.us.acme.com|localhost|127.0.0.1|myhost-pc

10 Click OK

CREATE A NEW BPEL PROJECT

Now you will use the JDeveloper BPEL Designer's New Project wizard, which automatically generates the skeleton of a BPEL project: the BPEL source, a WSDL interface, a BPEL deployment descriptor, and an Ant script for compiling and deploying the BPEL process.

To create your first BPEL project:

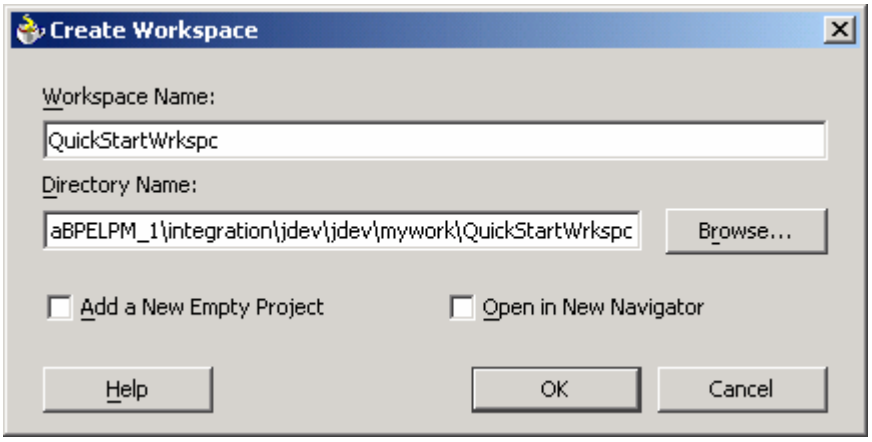
1 First you need to create a new workspace. This workspace will contain your BPEL projects.

2 In the **Applications – Navigator** right click **Applications** and select **New.**

In the wizard that appears select **Workspace** from the menu on the right, click **OK**.

3 In the next window that appears enter *QuickStartWrkspc* in the **Workspace Name** field.

Deselect the **Add New Empty Project** check box and click **OK**.



You have now created your BPEL development workspace you can view it in the **Application – Navigator** on the left.

Now you need to create a BPEL project within this workspace.

4 Right click on your workspace name and select **New Project**.

The New Project wizard appears.

5 Under the **Categories** column select **General > Projects.**

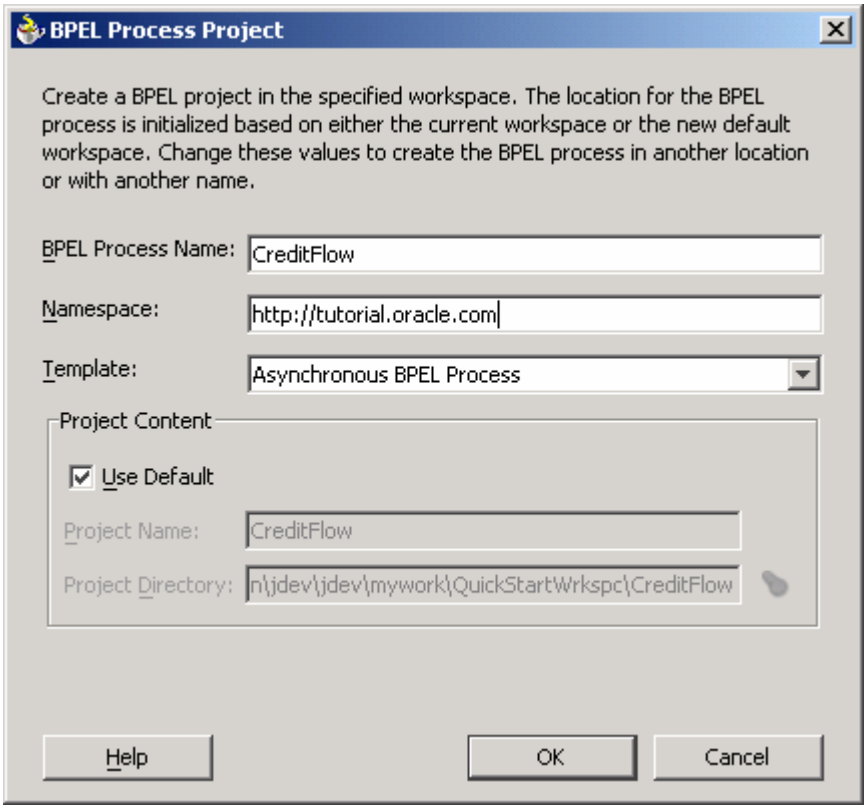
In the **Items** window on the right select **BPEL Process Project** and click **OK**.

The BPEL Process Project window appears.

6 Enter *CreditFlow* as the **BPEL Process Name**.

7 [Optional] Enter `http://tutorial.oracle.com` as the Namespace.

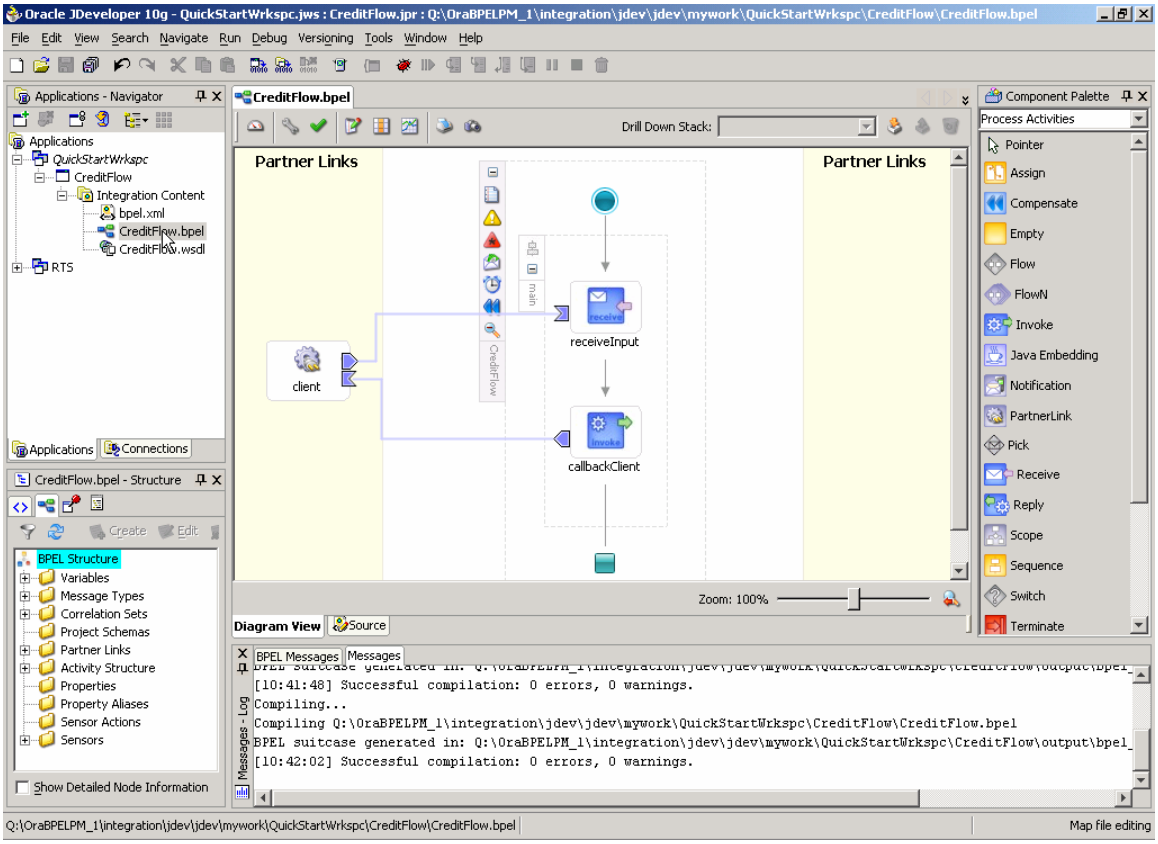
8 Leaving the template as **Asyncronous BPEL Process** and the **Use default** location at its default setting, click **OK**.



The new project will be created in the:

`C:/`*`Oracle_Home`*`/integration/jdev/jdev/mywork/QuickStartWrkspc/CreditFlow` directory. You can see (in the center pane within BPEL Designer) that the New Project wizard has created a skeleton for a new asynchronous BPEL process, with all the necessary source files. The file names are the same whether you create a synchronous or an asynchronous process, but the contents of the `.bpel` and `.wsdl` files will differ as appropriate. If you would like to go through a tutorial of creating a synchronous BPEL process, a simple HelloWorld tutorial is available at http://otn.oracle.com/bpel, which shows how to create a synchronous BPEL process with the Designer.

Note that JDeveloper is now showing 5 key panes, the center pane displays the BPEL process itself. The Navigator on the left displays workspaces, projects and project source files. The Structure pane on the lower left shows further information about the currently selected file in the center pane and there is a Message – Log on the bottom of the screen that displays the status of the project. The final pane is the Component Palette on the right of screen that displays the process activities available to drag and drop into your BPEL process.



REVIEW THE WSDL INTERFACE OF YOUR ASYNCHRONOUS BPEL PROCESS

You will now review (and, in the next section, edit) the WSDL file for the process.

To view the WSDL interface:

1 In the Navigator, double-click the `CreditFlow.wsdl` file to edit it.

2 Scroll down as necessary to browse the code. The most “interesting” parts are pointed out below.

Notice that the New Project wizard has defined both a `CreditFlowRequest` complex-type element that your flow accepts as input (in a document-literal style WSDL message) and a `CreditFlowResponse` element that it returns.

Because this process is asynchronous, two portTypes are defined, each with a one-way operation: one to initiate the asynchronous process and one for the process to call back the client with the asynchronous response.

```
<!-- portType implemented by the CreditFlow BPEL process -->
<portType name="CreditFlow">
    <operation name="initiate">
        <input message ="client:CreditFlowRequestMessage"/>
    </operation>
</portType>

<!-- portType implemented by the requester of CreditFlow BPEL process
     for asynchronous callback purposes -->
<portType name="CreditFlowCallback">
    <operation name="onResult">
        <input message ="client:CreditFlowResponseMessage"/>
    </operation>
</portType>
```

Also note that the partnerLinkType defined for this asynchronous process has two roles, one for the service provider and one for the requester.

```
<plnk:partnerLinkType name="CreditFlow">
    <plnk:role name="CreditFlowProvider">
        <plnk:portType name="client:CreditFlow"/>
    </plnk:role>
    <plnk:role name="CreditFlowRequester">
        <plnk:portType name="client:CreditFlowCallback"/>
    </plnk:role>
</plnk:partnerLinkType>
```

EDIT THE WSDL INTERFACE OF YOUR BPEL PROCESS

Now you will edit the WSDL file to modify the input and output messages.

To modify the input and output messages of your BPEL process:

1 While still editing your `CreditFlow.wsdl` file, change the two type definitions so that your flow will take an `ssn` field as input (keeping the `string` type) and return a `creditRating` element as output of type `int`. The parts you need to change are shown in bold (and red) below.

```
<element name="CreditFlowProcessRequest">
    <complexType>
        <sequence>
            <element name="ssn" type="string">
        </sequence>
    </complexType>
</element>

<element name="CreditFlowProcessResponse">
    <complexType>
        <sequence>
            <element name="creditRating" type="int">
        </sequence>
    </complexType>
</element>
```

2 Save your WSDL file and close the window in which you have been editing it.

Note that instead of using a text editor to change the WSDL file, you could use any XML Schema or WSDL editing tool that you choose. JDeveloper version 10.1.3 will bundle such a tool and several are also available for the Eclipse platform or as standalone tools.

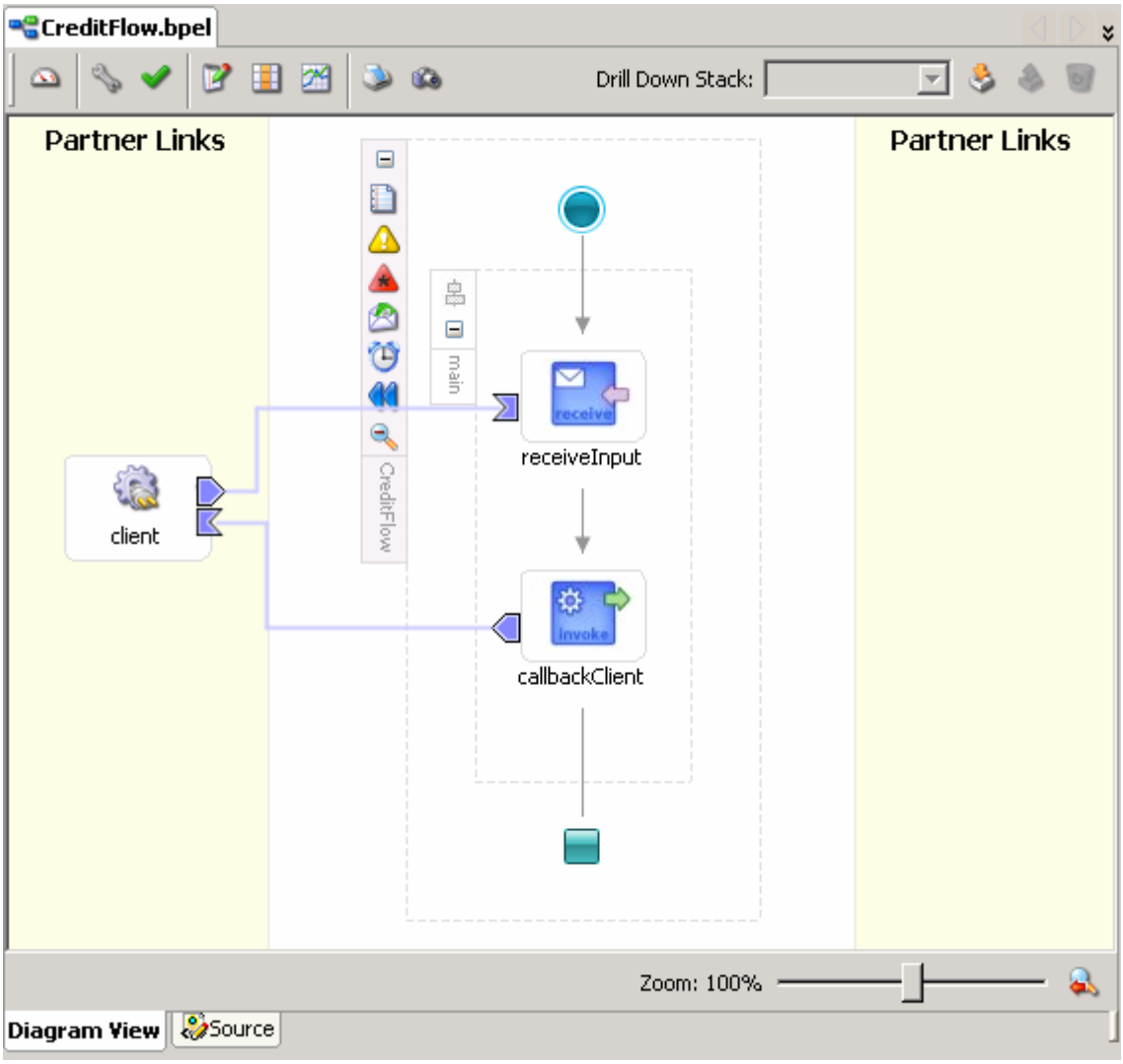
SWITCH BETWEEN THE DIAGRAM VIEW AND SOURCE CODE

You will now turn to the BPEL file and view it in a variety of ways.

To view the BPEL process:

3 Double-click `CreditFlow.bpel` in the Navigator if it is not already open in the Designer (however, it should be already open if you have followed this tutorial exactly).

The initial view that you will see for editing a BPEL process file consists of 3 swim-lanes; the colored outer swim-lanes contain references to external services called Partnerlinks. The center swim-lane contains the CreditFlow process itself. Note that the BPEL Designer automatically puts a partnerLink named `client` on the left side of the window. This represents any process, service or interaction that initiates your CreditFlow service.



REVIEW THE BPEL SOURCE CODE

The New Project wizard has created a basic skeleton for you of an asynchronous BPEL process.

To view the BPEL source code:

1 With `CreditFlow.bpel` open and active, click the **Source** tab at the bottom of the `CreditFlow.bpel` window.

2 Scroll down as necessary to browse the code. The interesting parts are pointed out below.

The partnerLink created for the client interface includes two roles, `myRole` and `partnerRole` assignment. As you saw in the WSDL file for this process, an asynchronous BPEL process typically has two roles for the client interface: one for the flow itself, which exposes an input operation, and one for the client, which will get called back asynchronously.

```
<partnerLinks>
    <!-- comments... -->
    <partnerLink name="client"
                 partnerLinkType="client:CreditFlow"
                 myRole="CreditFlowProvider"
                 partnerRole="CreditFlowRequester"
                 />
</partnerLinks>
```

Also, the `<receive>` activity in the main body of the process is followed by an `<invoke>` activity to perform an asynchronous callback to the requester. (Note the difference between this and a synchronous process, which would use a `<reply>` activity to respond synchronously to the caller.)

```
<sequence name="main">

    <!-- Receive input from requester.
         ...
         -->
    <receive name="receiveInput" partnerLink="client"
             portType="client:CreditFlow"
             operation="initiate" variable="ssn"
             createInstance="yes"/>

    <!-- Asynchronous callback to the requester.
         ...
         -->
    <invoke name="callbackClient"
            partnerLink="client"
            portType="client:CreditFlowCallback"
            operation="onResult"
            inputVariable="creditRating"
            />
</sequence>
```

The process is editable from both the source view and the diagram view interchangeably.

Switch back to visual design view:

1 With `CreditFlow.bpel` open and active, click the **DiagramView** tab at the bottom of the `CreditFlow.bpel` window to return to the Diagram view.

ADD ACTIVITIES TO THE PROCESS MAP

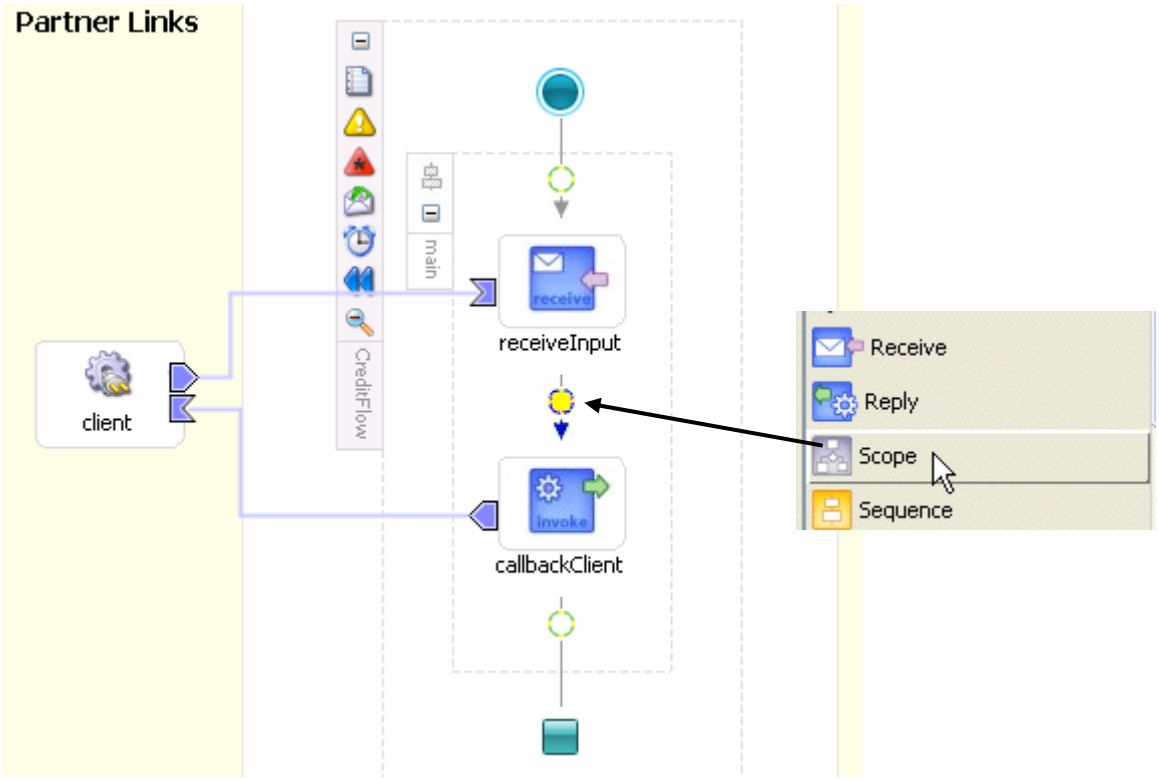
You are now ready to edit the process.

To insert a <scope> activity:

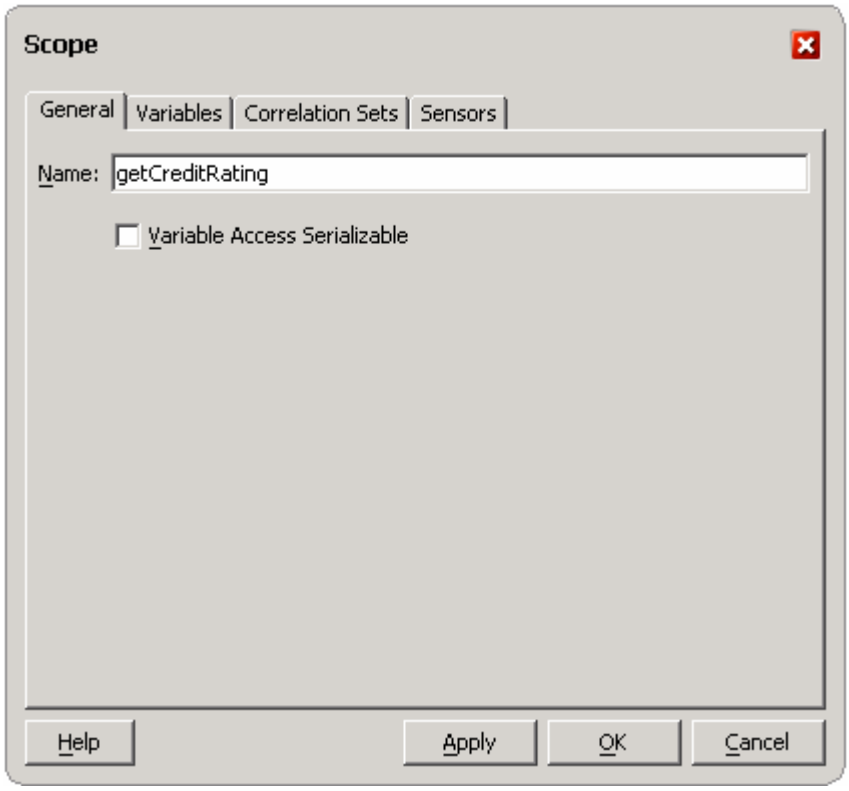
A BPEL scope is a collection of activities that can have its own local variables, exception handling, compensation, and so on — very much analogous to a programming language `{ }` block.

1 In the BPEL Component Palette on the right, make sure Process Activities is selected in the drop down.

2 Drag a `<scope>` activity (from **scope** on the BPEL Palette) to the available position between the `receiveInput <receive>` activity and the `callback`Client `<invoke>` callback element.



3 Double click the scope element and enter *getCreditRating* into the name field.



To insert an <invoke> activity into the scope:

4 Click the "+" icon to the left of the `<scope>` activity in the process flow, to expand the scope so that you can drop activities into it.

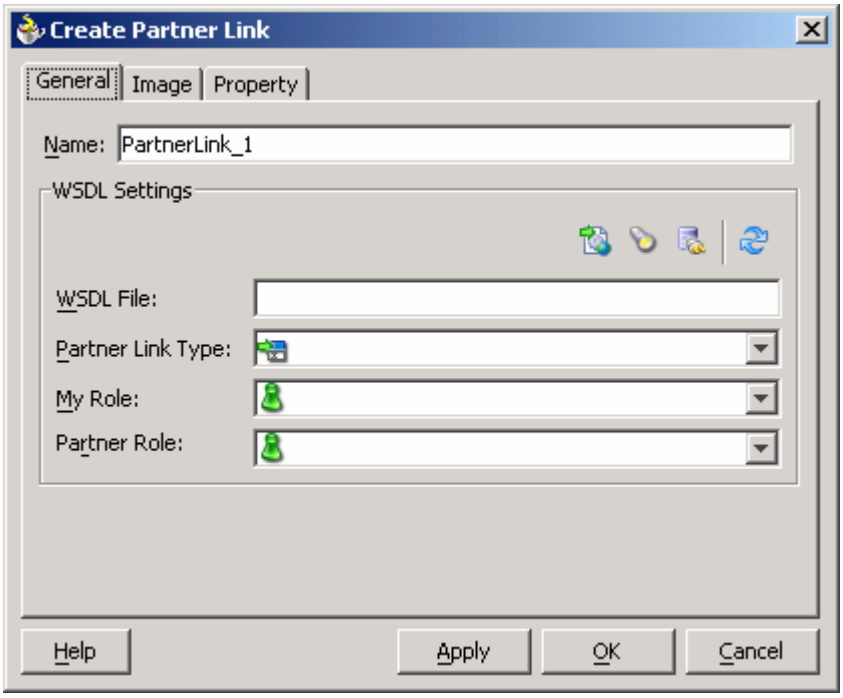
5 Drag an `<invoke>` activity from the BPEL Palette into the **Drop activity here** area within the scope.

The next step is to configure the `<invoke>` activity to call your intended service (in this case the credit rating service). In BPEL, this means you first need to create a partnerLink for the service.

CREATE A PARTNERLINK FOR THE CREDIT RATING SERVICE

Partnerlinks represent services that BPEL processes can call or interact with.

1 In the component palette drag and drop the partnerlink activity into the right swimlane.



This will bring up the Create PartnerLink Wizard above.

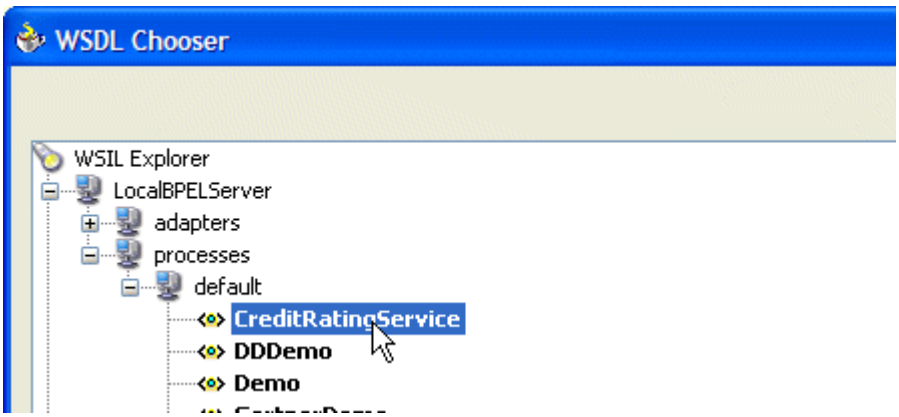
2 Enter the name `creditRatingService` for the partnerLink.

3 Select the flashlight icon above the WSDL File text area.

This opens the service browser where you can select the service endpoint you are looking for.

4 The wizard presents an explorer listing all the services deployed on the local Oracle BPEL server (as well as other directories that you configure it for).

Navigate to **LocalBPELServer > processes > default > CreditRatingService**.



5 Click **OK**.

You have selected the CreditRatingService you deployed to the local BPEL server earlier with the obant command.

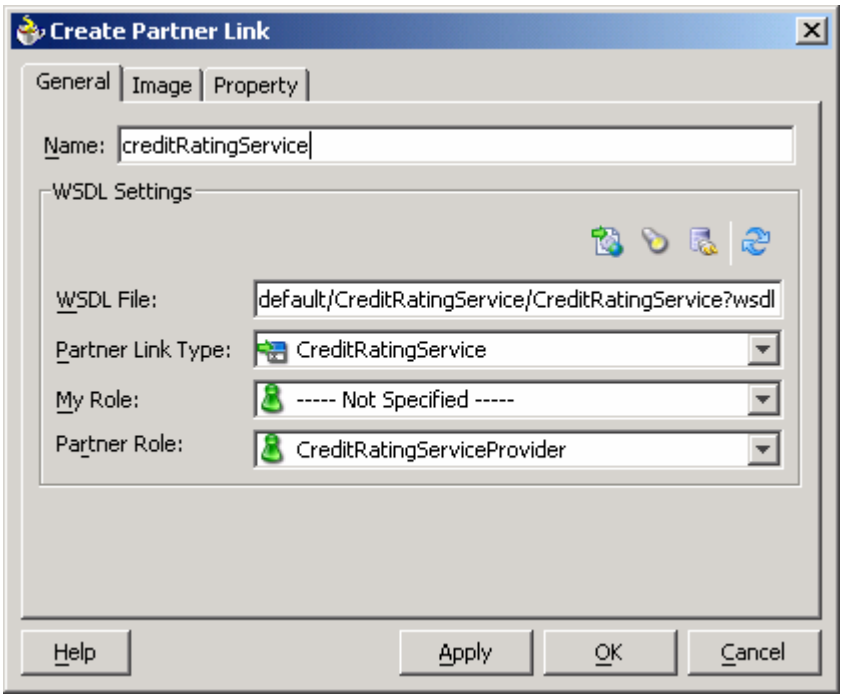
6 The browser will close and the URL of the service will appear in the WSDL File field. The wizard will fetch the contents of the WSDL file so that it can populate the drop-down lists appropriately.

Alternatively, you can enter the URL for a WSDL directly in this field if you know the specific location of the WSDL for the service you want to invoke.

7 In the Partner Role list, select `CreditRatingServiceProvider`.

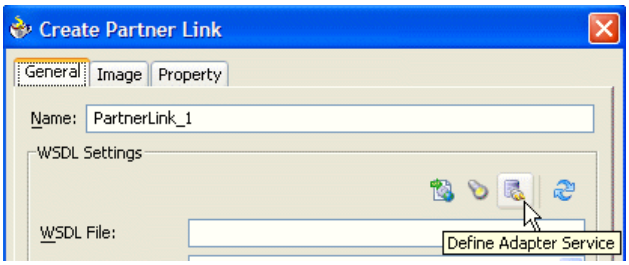
You will leave the `myRole` field blank. (Because this is a synchronous service without any callbacks, the client does not need a role.)

8 Click **Apply**, then **OK**.

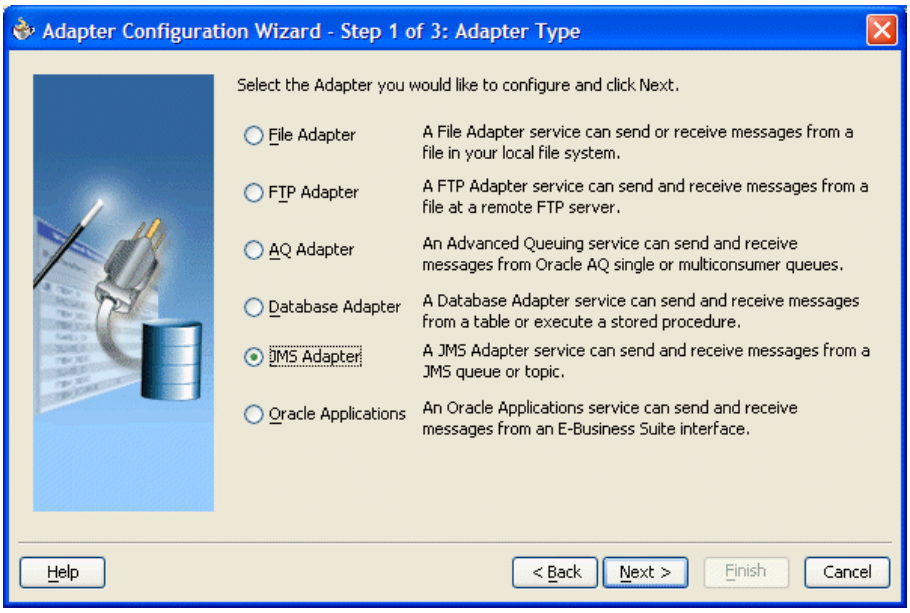


A new partnerLink has been added to your flow.

As described previously, you can also use adapters to connect to many different back-end systems which typically do not have Web services interfaces. Adapters are available for hundreds of systems and protocols and wizards bundled into JDev for several of them. To bring up the adapter wizards, select the database icon in the Create Partner Link dialog:



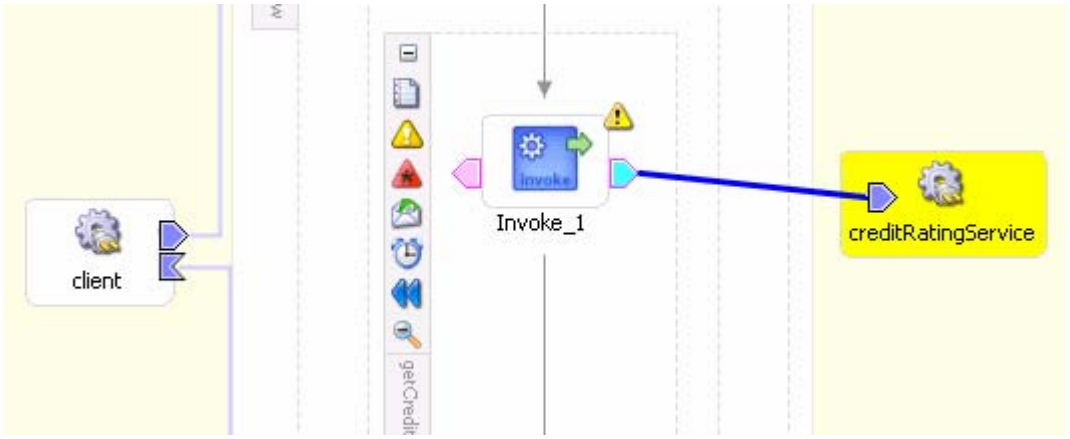
You can then see the bundled adapter wizards for things like reading/writing files with non-XML formats, FTP interactions, JMS messages, database operations, etc.



CONFIGURE THE <INVOKE> ACTIVITY

Now you will need to associate the invoke activity with the partnerlink you just created.

1 The easiest way to do this is to select the blue arrow on the right of the invoke activity and drag it onto the creditRatingService partnerlink.



After taking this step the Edit Invoke wizard appears.

The Partnerlink Field is already filled out. The Operation drop-down box has the operation process selected, the creditRatingService only provides one available operation.

2 Type *invoke_CRS* in the name field.

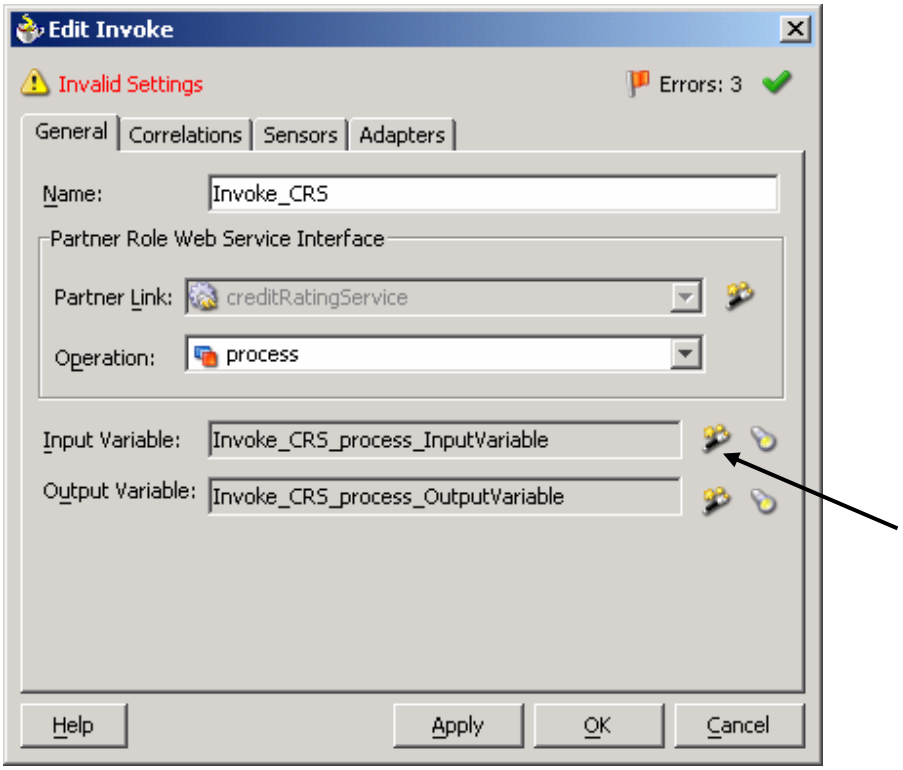
You need to specify an input variable and an output variable. These two variables are passed to, and received from the creditRatingService.

3 The wizard allows you to create a new variable or select existing variables.

You will create new variables by selecting the wand icon on the right of the variable fields.

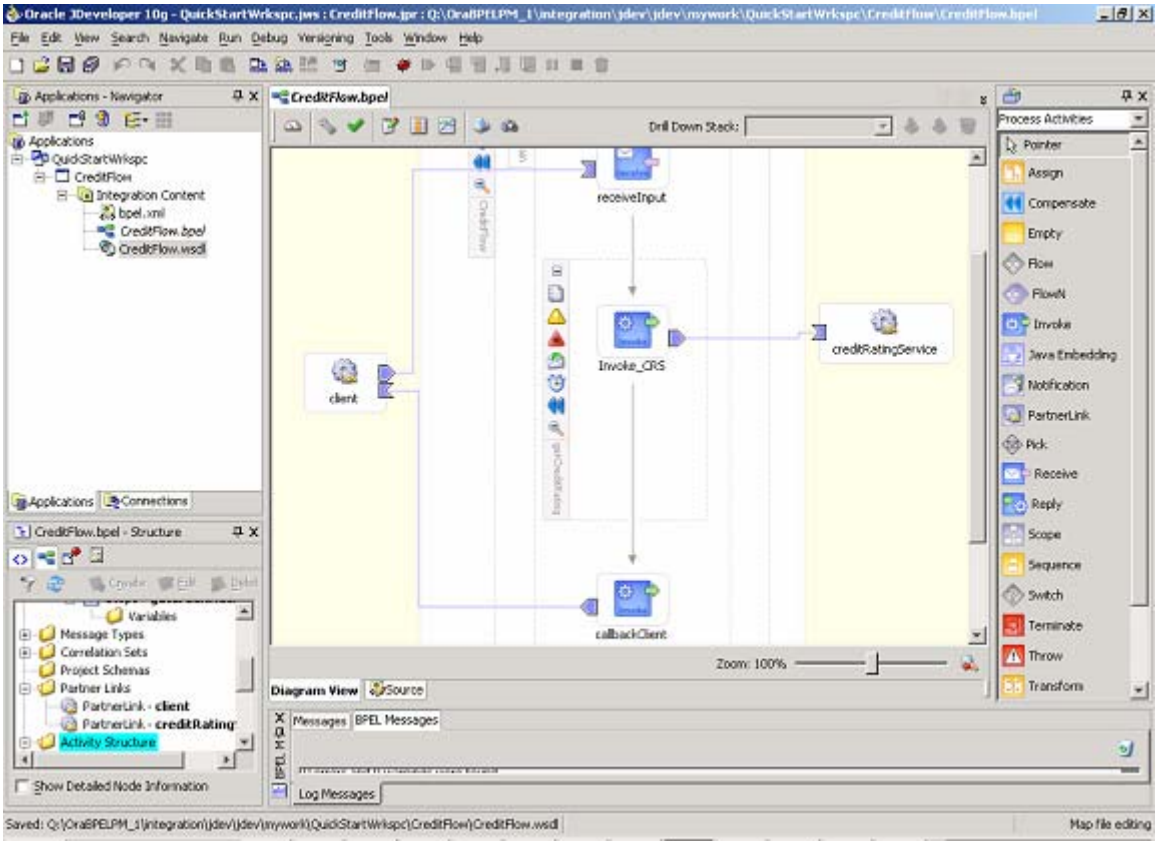
A new Create Variable wizard appears, accept the defaults (or specify your preferred variable name) and select **OK**.

Complete this step for both the Input and Output variables.



4 Select **Apply** and **OK**.

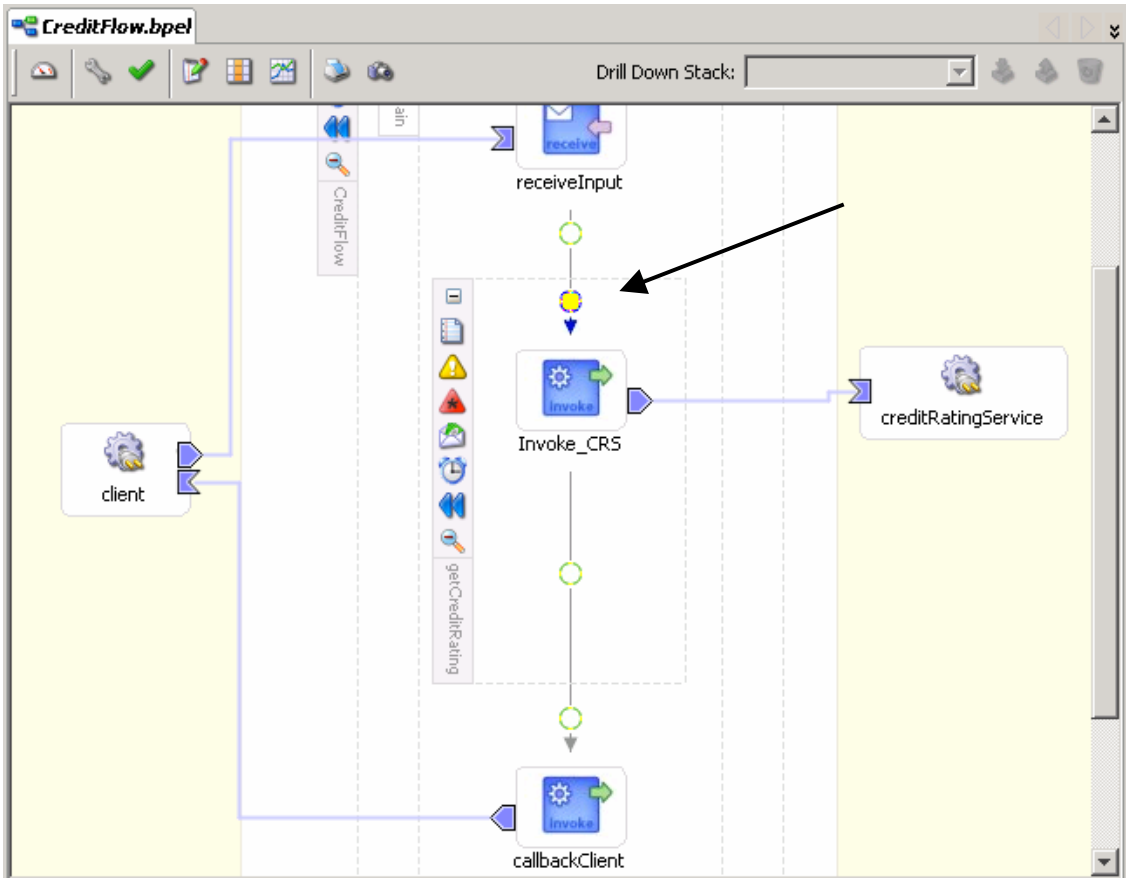
5 You have now configured the Invoke activity to call the creditRatingService partnerlink.



INITIALIZE THE INPUT VARIABLE

Now you will use XPath and the BPEL `<assign>` activity to perform simple data manipulation to initialize the input variable you are passing to the credit rating service.

1 Drag an `<assign>` activity from the BPEL Palette into your flow just before the invocation of the credit rating service (but within the `getCreditRating` scope).



2 Double click the newly created Assign activity to update its properties.

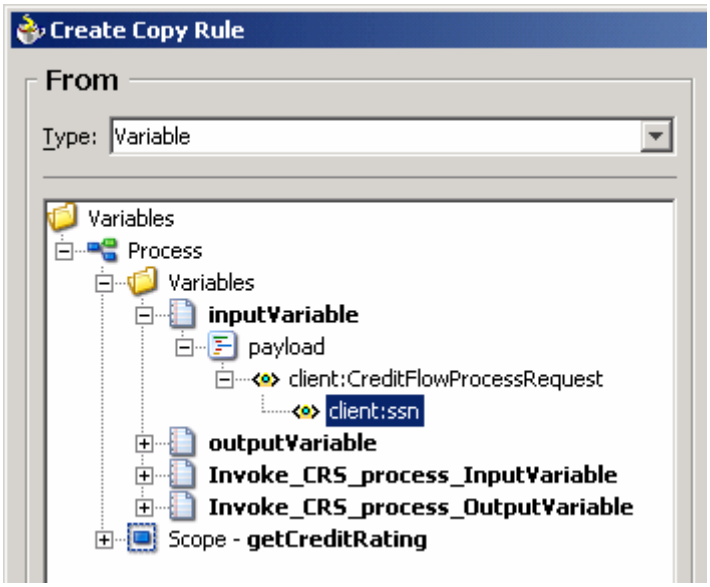
3 Select the general tab on the Assign activity and enter *assign_SSN* into the Name field.

4 Select the Copy Rules tab and click **Create**.

The XPath wizard opens. This wizard allows you to assign the value of a variable selected in the From field to the value of a variable selected in the To field.

5 In the From section of the Copy Rule form ensure **Variable** is selected in the Type drop-down box and navigate to and select

Variables > inputVariable > payload > client:CreditFlowProcessRequest > client:ssn



6 Fill in the **To** section of the Copy Rule form as follows:

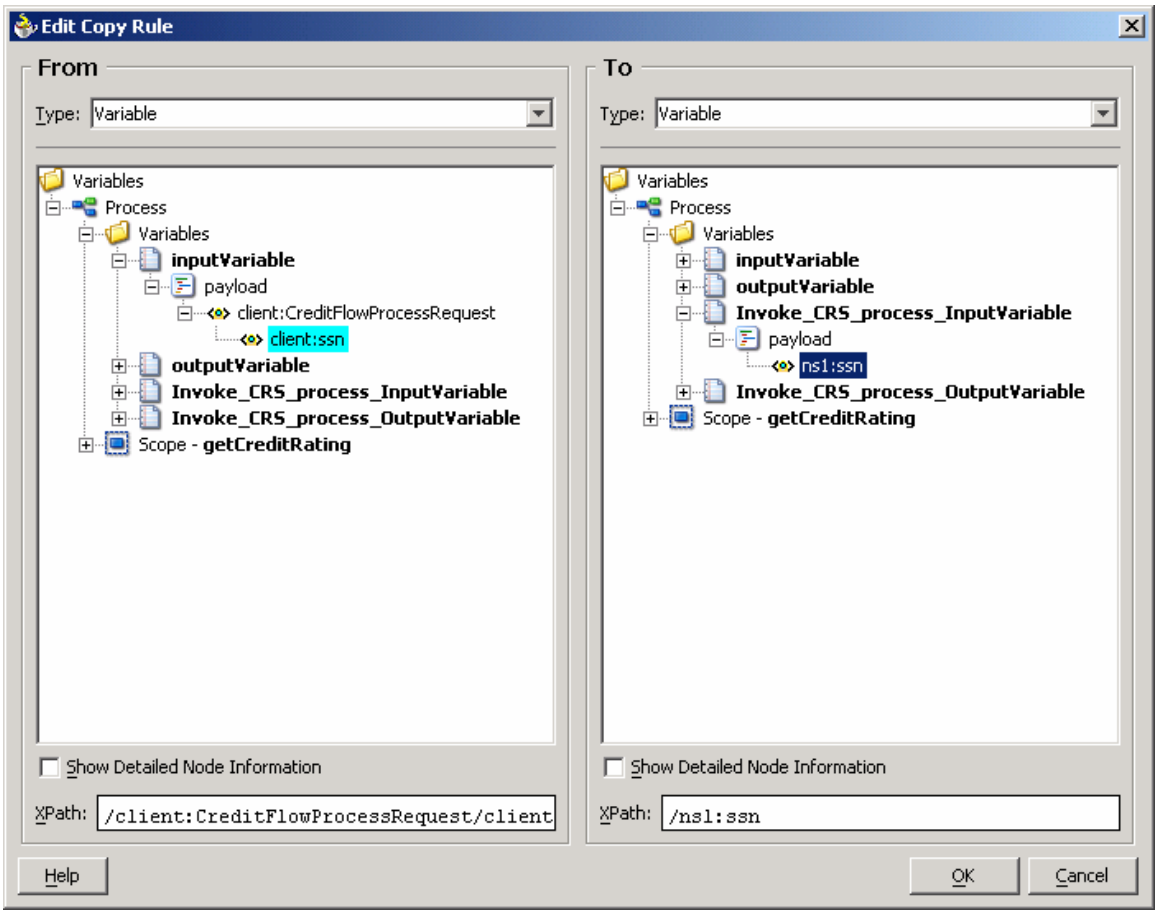
   a. In the Type drop-down select **Variable**.

   b. Navigate to and select

      Variables > Invoke_CRS_process_InputVariable > payload > > ns1:ssn

7 In the Copy Rule form, click **OK**.

8 In the Assign form select **Apply**, then **OK**.



Note that you can always enter XPath queries directly into the text field if you know what the correct query is. For more information regarding data manipulation in BPEL please refer to the BPEL Data Manipulation chapter in the BPEL PM Developer's Guide at http://otn.oracle.com/bpel.

COMPILE, DEPLOY, AND TEST YOUR BPEL PROCESS

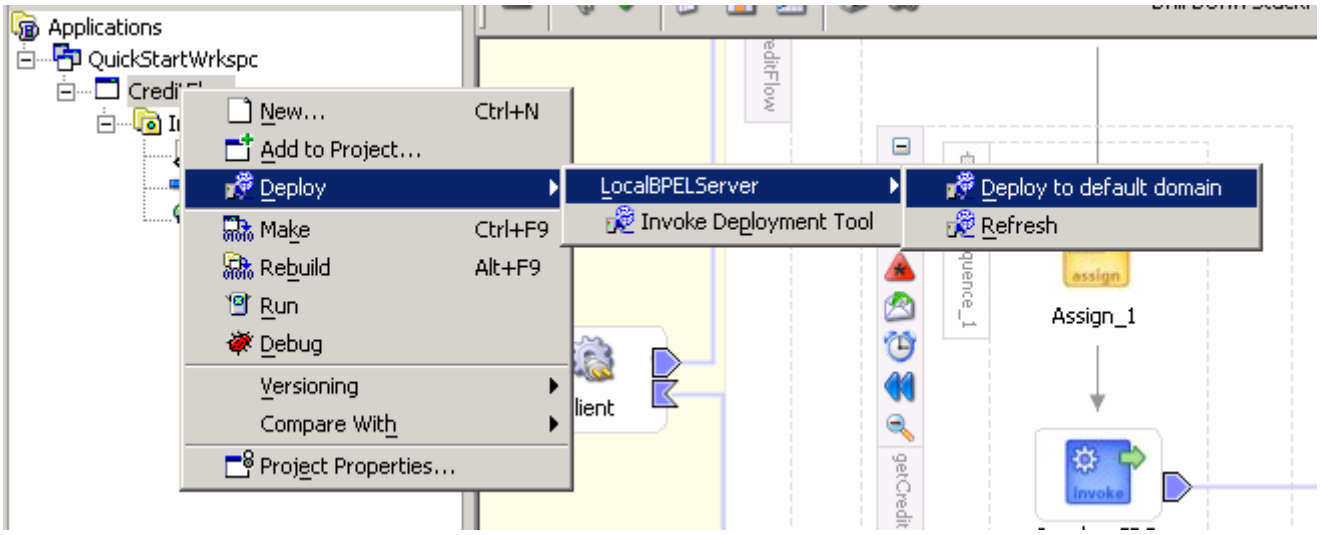
Although you have not yet wired up the return value from the credit rating service to the return value of your flow, you can still test your flow. In this section of the tutorial, you will compile, deploy, and test your BPEL process.

To compile and deploy your BPEL process:

1 Save the process.

In the toolbar select **Save All**.

2 To compile and deploy your BPEL process right click your project (CreditFlow) and select **Deploy > LocalBPELServer > Deploy to default domain.**



If this is the first project you have installed you may need to select **Refresh** before you can see **Deploy to default domain**.

A connection to the default domain on the local BPEL server is created by the install process, you can view it in the connections tab on the Navigator window.

3 Enter the default domain password at the prompt.

By default the password is *bpel*.

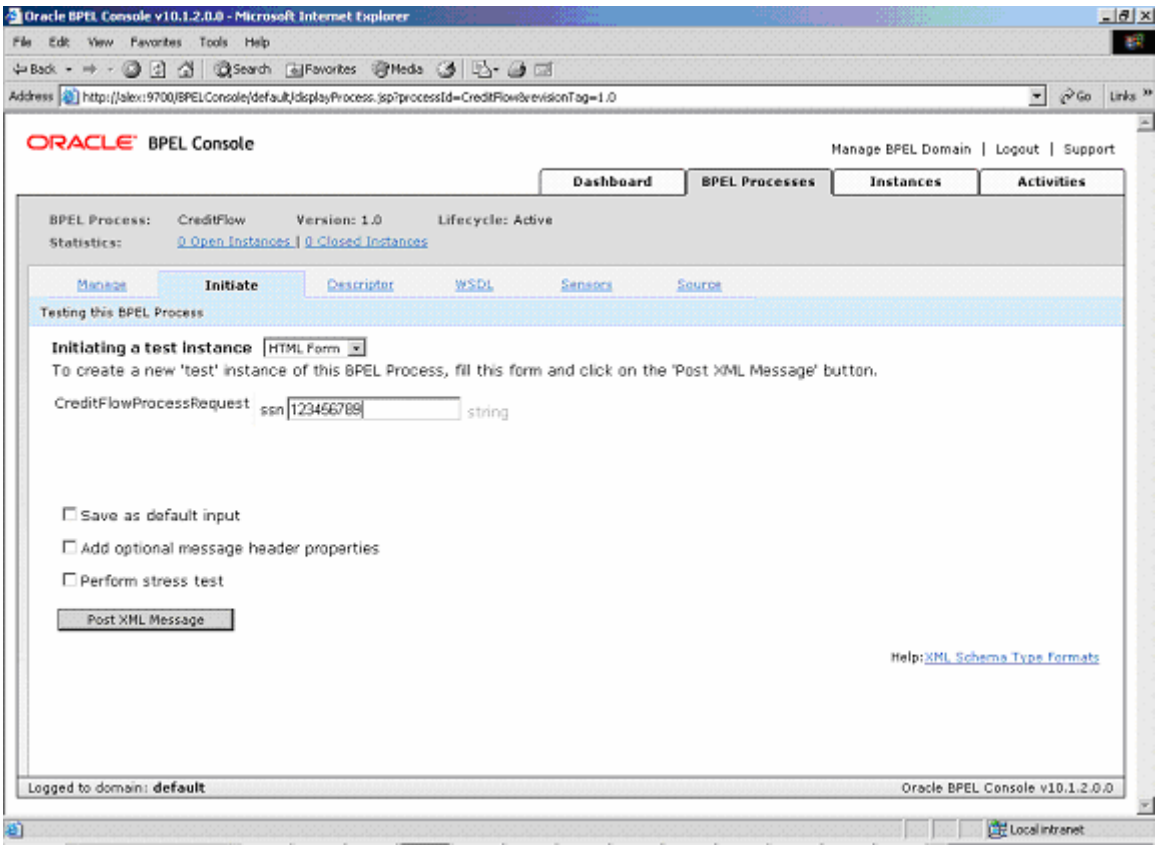
As the final step, you will test your BPEL process by using the automatically generated test interface in the BPEL Console (similar to what you did at the beginning of this tutorial, if you tested the deployed credit rating service).

To initiate a test instance of your BPEL process:

4 Bring the BPEL Console into view.

The BPEL Console can be found at http://localhost:9700/BPELConsole.

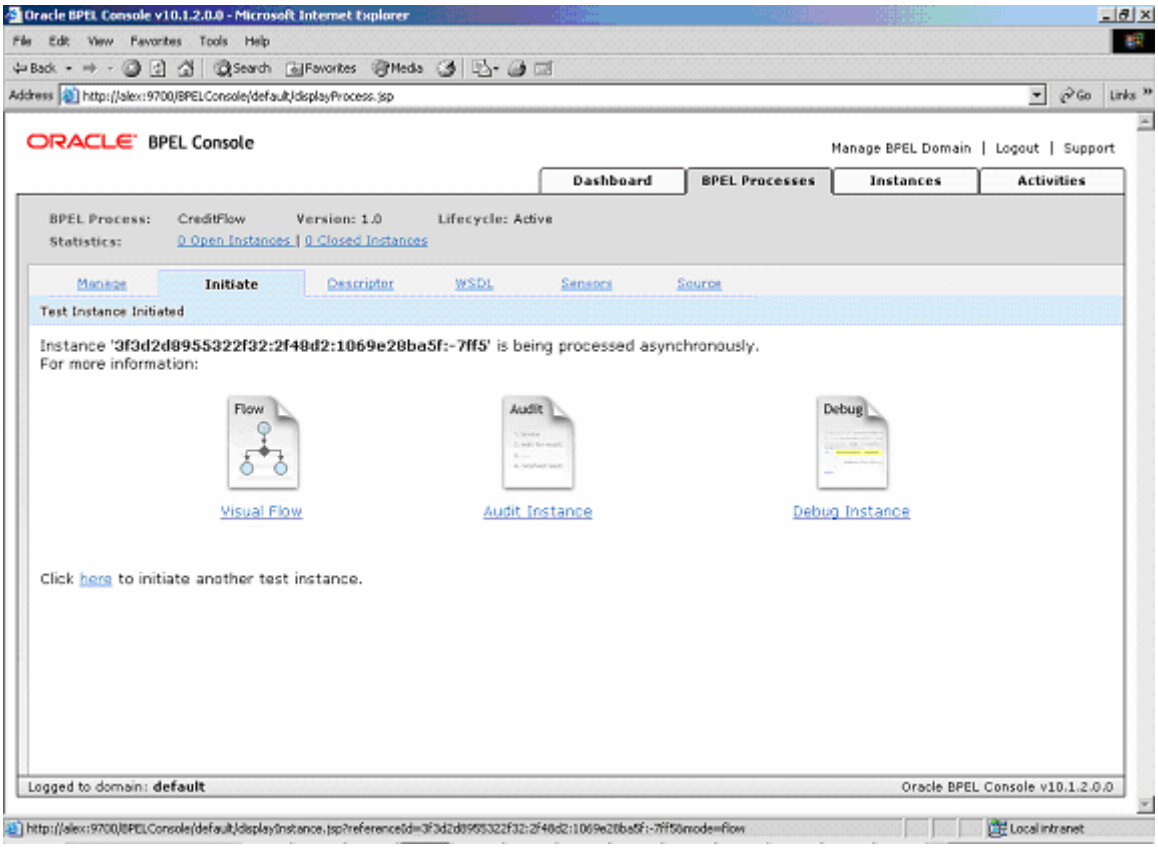
5 Click `CreditFlow` on the Dashboard and in the automatically generated HTML Form interface that appears, enter a nine-digit number that does *not* begin with a 0, and click **Post XML Message** to initiate the process.



To view the visual audit trail of the instance:

6 Click the **Visual Flow** link to see a visual audit trail representing the current state of the process instance.

Note that because this process is asynchronous and long-running you do not get the result returned right away (like you did when testing the CreditRatingServce).



Upon selecting the Visual Flow link, you will see an audit trail displaying the current state of the process, very similar to the process map displayed by the BPEL Designer. It will show that you have successfully invoked the credit rating service.

7 Click the `Invoke_CRS` `<invoke>` activity in the audit trail (a few boxes down) to see the messages sent to and received from the credit rating service.

To complete the implementation of this flow, you would add another `<assign>` activity to the flow (after the credit rating service has been invoked), which would copy the result returned from the credit rating service (in your `invoke_CRS_process_output_variable` variable) into the `creditRating` field for the result of your process itself (the `outputVariable` variable). We leave this as an exercise for you.

If you experience any difficulties in completing the flow or have any questions or comments regarding this tutorial, please use the OTN BPEL forum at http://otn.oracle.com/bpel.

Example II: A Loan Procurement BPEL Process

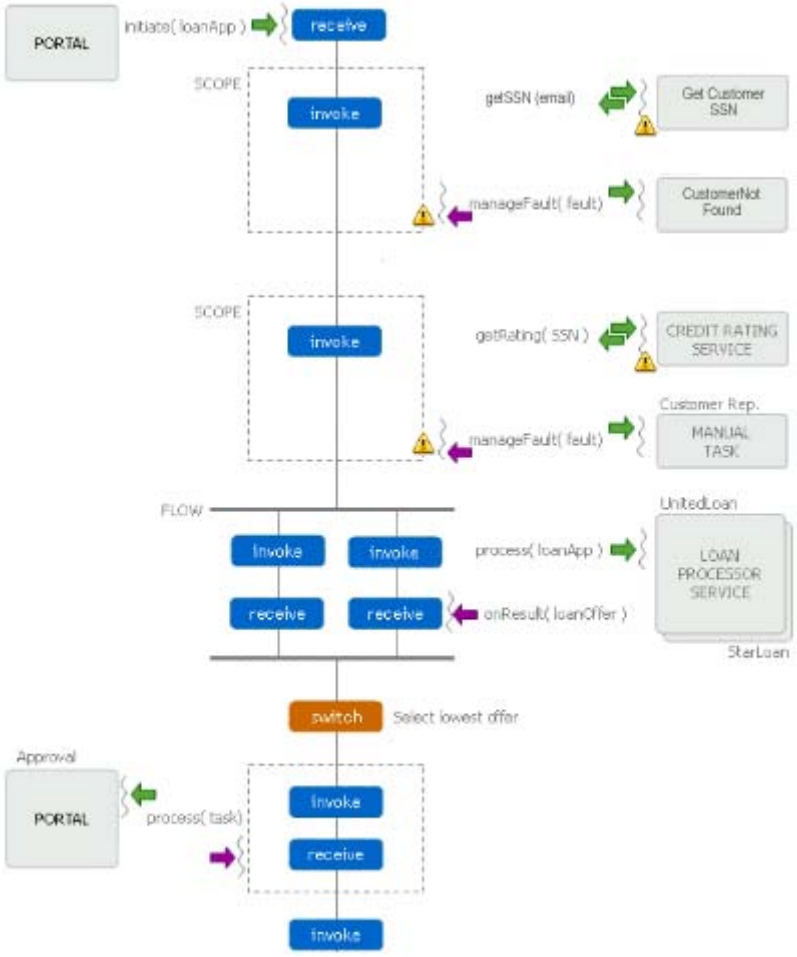
In this example you examine how a more sophisticated BPEL process can be implemented and executed using the Oracle BPEL Process Manager.

THE LOAN FLOW PLUS PROCESS

Most business processes will be much more complex than the simple example you created in the previous section. Fault-handling, interaction with asynchronous services, user/manual tasks, Java/J2EE code integration and custom user interfaces/portals are all key requirements for many business flows. Here you will see how you would build, test, debug and manage a BPEL process that implements all of these requirements.

The process is called “LoanFlowPlus” and is shipped with the samples for any Oracle BPEL Process Manager installation in the directory:
\integration\orabpel\samples\demos\LoanDemoPlus

The requirements implemented by this application are shown in the diagram below:



This process is a long-running BPEL flow which takes a LoanApplication document as input and then will asynchronously return a selected and approved loan offer. At the

beginning of the execution of the process, it fetches a social security number for the customer from an EJB (accessed as a service through the native EJB WSIF binding) and then go out to the synchronous CreditRating service that you saw previously to request a credit rating for the customer. The process is coded to handle the NegativeCredit faults that may be thrown by the credit rating service if a negative credit event is seen for the customer (for example a bankruptcy). If such an exception is thrown, the LoanFlowPlus process will queue up a task for a customer service rep to manually handle the exception and then wait for the task to complete. The customer service rep can use a dashboard UI to see their tasks and specify a credit rating or choose to cancel each application.

Once a credit rating is supplied, either through manual or automated activities, the process will fall through into automated processing. It will then send the completed loan application to two loan processors, who can each take an arbitrary amount of time before returning loan offers. Since the loan processors can be long-running, they are implemented as asynchronous services and the BPEL process will invoke them in parallel.

In the case of this flow, it waits for both loan offers to be received before using some simple branching logic to select the best offer (the one with the lowest interest rate). Finally, another user task allows the customer to review and approve the selected loan offer. Once this task is completed, the selected and approved offer is returned as the result of the flow.

REVIEW THE PROCESS IN THE BPEL DESIGNER

Build and deploy external dependencies

As in the previous section, the LoanFlowPlus process is dependent on several external services which you can quickly build and deploy using the 'obant' command. To do this:

1 Open up a command prompt if you do not already have one open.

2 Change to the `orabpel\samples\demos\LoanDemoPlus` directory as follows:

```
> cd Oracle_home\integration\orabpel\samples\demos\LoanDemoPlus
```

3 Execute the **obant** command.

```
> obant
```

This will build and deploy the loan services and other pre-requisites required for the LoanFlowPlus process, as well as the process itself.

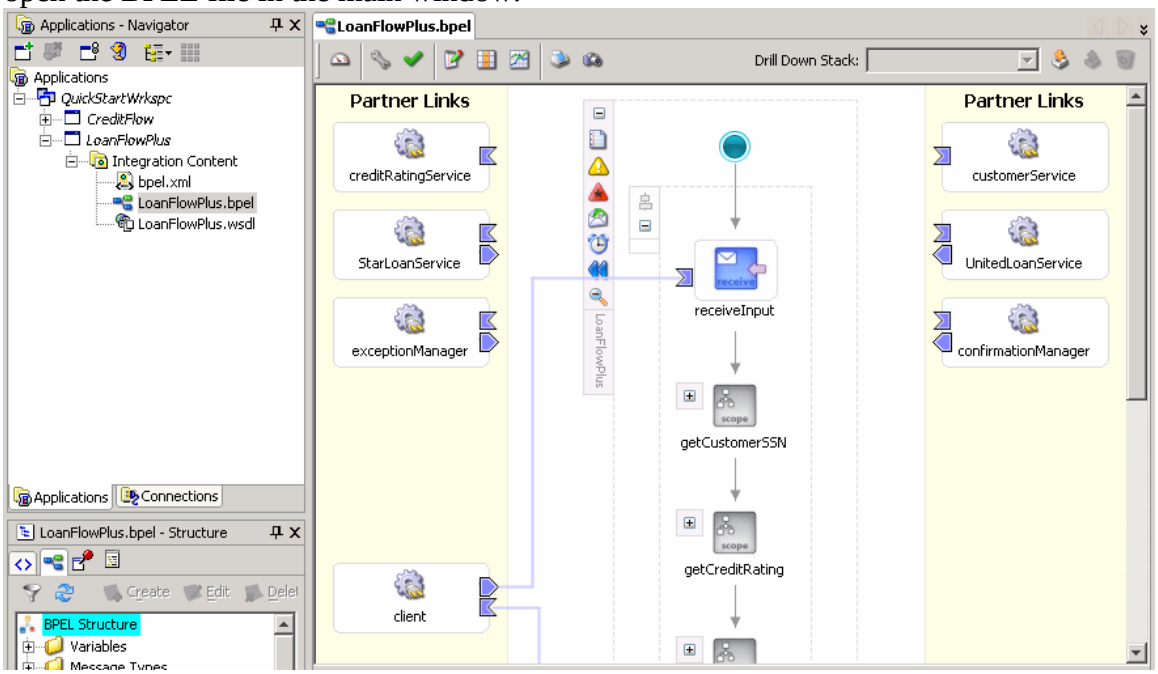
Launch the BPEL Designer

4 You can now use the BPEL Designer to review the BPEL implementation of the LoanFlowPlus process. To do this, start up the Designer as you did in the previous section, and ensure the *QuickStartWrkspc* you created earlier is highlighted.

Then select **File > Open** and in the explorer window that appears navigate to ***Oracle_home*\integration\orabpel\samples\demos\LoanDemoPlus\LoanFlowPlus** and select the **LoanFlowPlus.jpr** file. This is the LoanFlowPlus project file.

5 Select **Open**. The LoanFlowPlus project opens in the Navigator.

6 Open LoanFlowPlus > Integration Content and double-click LoanFlowPlus.bpel to open the BPEL file in the main window.



You can see that the creditRatingService is the same synchronous service used in the previous section, however the StarLoan and UnitedLoan services are asynchronous, supporting a one-way initiate operation and an asynchronous onResult callback which will be called after a loan offer is ready from each service. That could take anywhere from a few seconds to a few days or more. As with your previous process, the client interface to the LoanFlowPlus process itself is also asynchronous.

7 In the center swim-lane of the Designer you will see a visual representation of the whole process. Keep in mind that the BPEL Designer uses standard BPEL source as its native format so you can always select the BPEL Source view to see the underlying BPEL source code and/or make changes in either view and see those changes represented immediately in the alternate view. Here you will see that the BPEL Designer view shows that the process gets initiated, then encapsulates the logic to fetch the ssn (including the exception handling code) in a BPEL <scope>, etc.

In the next few sections of this document, you will review the code which implements this process.

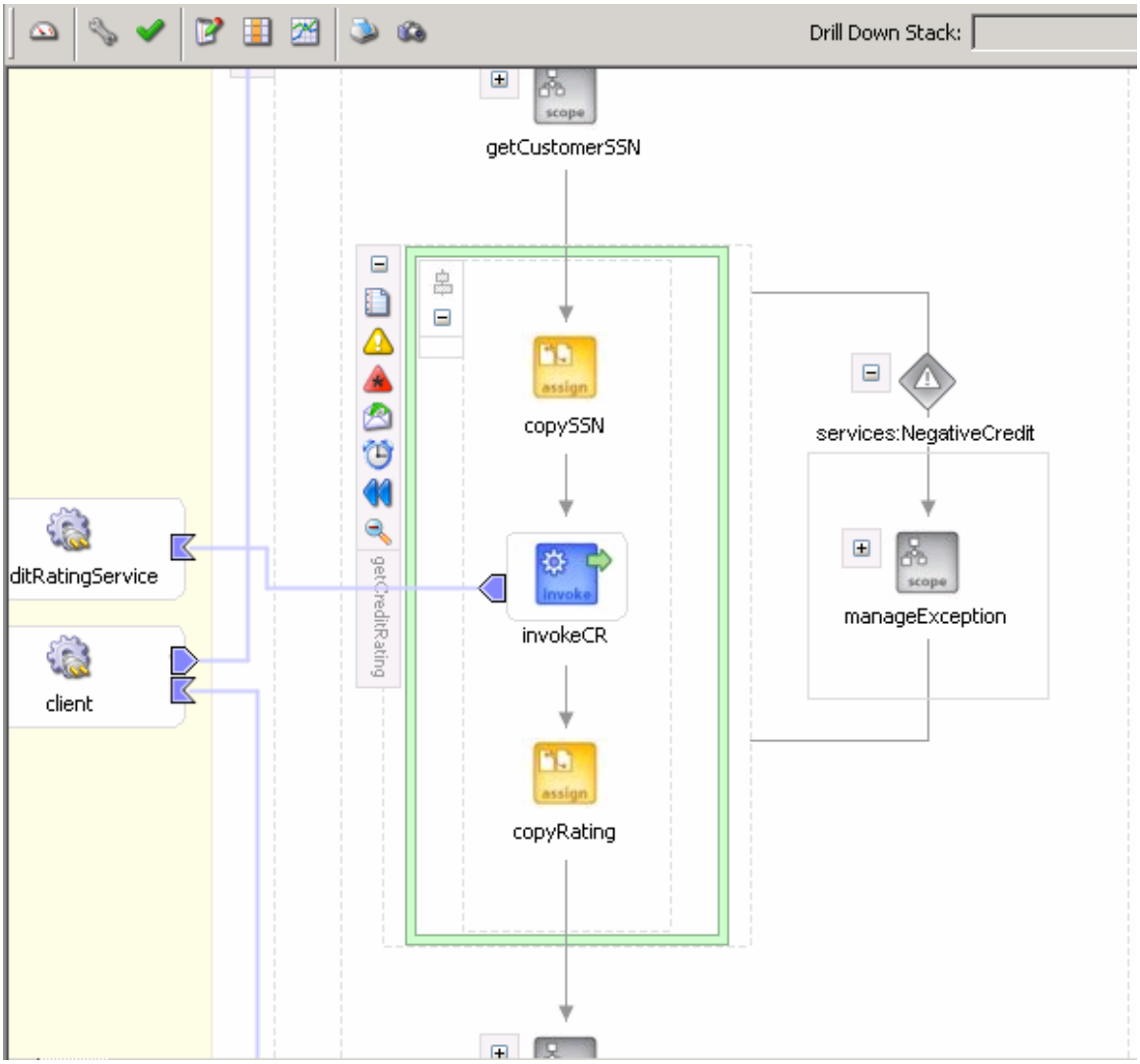
MANAGE FAULTS

The LoanFlowPlus process uses the BPEL fault handling mechanism to catch and manage exceptions thrown by the ssn and credit rating services. Fault handlers are associated with a particular <scope> activity and faults encountered within a scope and

not handled by that scope are bubbled up to the enclosing scope, just like try-catch in Java.

8 To see the NegativeCredit faultHandler defined for the getCreditRating scope, select the "+" to the left of the getCreditRating scope to open up the view of that scope.

9 When the scope opens you can view the main flow running top-to-bottom within the scope. You can also see a secondary flow as an extension out to the right, the top of this flow contains the error activity *services:NegativeCredit*.



You can open the scope activity within the exception flow and you will see the steps that are executed as the logic for handling this fault. In BPEL any activity (which can be

arbitrarily complex, short or long-running, etc) can be used in a faultHandler. In this case a user task is defined for a customer service rep to handle the exception in a manual fashion. Of course faults can also be handled via automated logic or re-thrown.

We won't look at the details of the BPEL source which implements this fault handler, but a skeleton of the code is shown below.

```
<scope name="getCreditRating" variableAccessSerializable="no">
    <variables>
        ...
    </variables>
    <!-- Watch for faults (exceptions) being thrown from
         creditRatingService -->
    <faultHandlers>
        <catch faultName="services:NegativeCredit"
               faultVariable="crError">
                <!-- Actual fault handling code goes here -->
                ...
        </catch>
    </faultHandlers>
    <sequence>
        <!-- Code during which you want to watch for faults -->
        ...
        <invoke name="invokeCR" partnerLink="creditRatingService"
                                portType="services:CreditRatingService"
                                operation="process"
                                inputVariable="crInput"
                                outputVariable="crOutput"/>
        ...
    </sequence>
</scope>
```

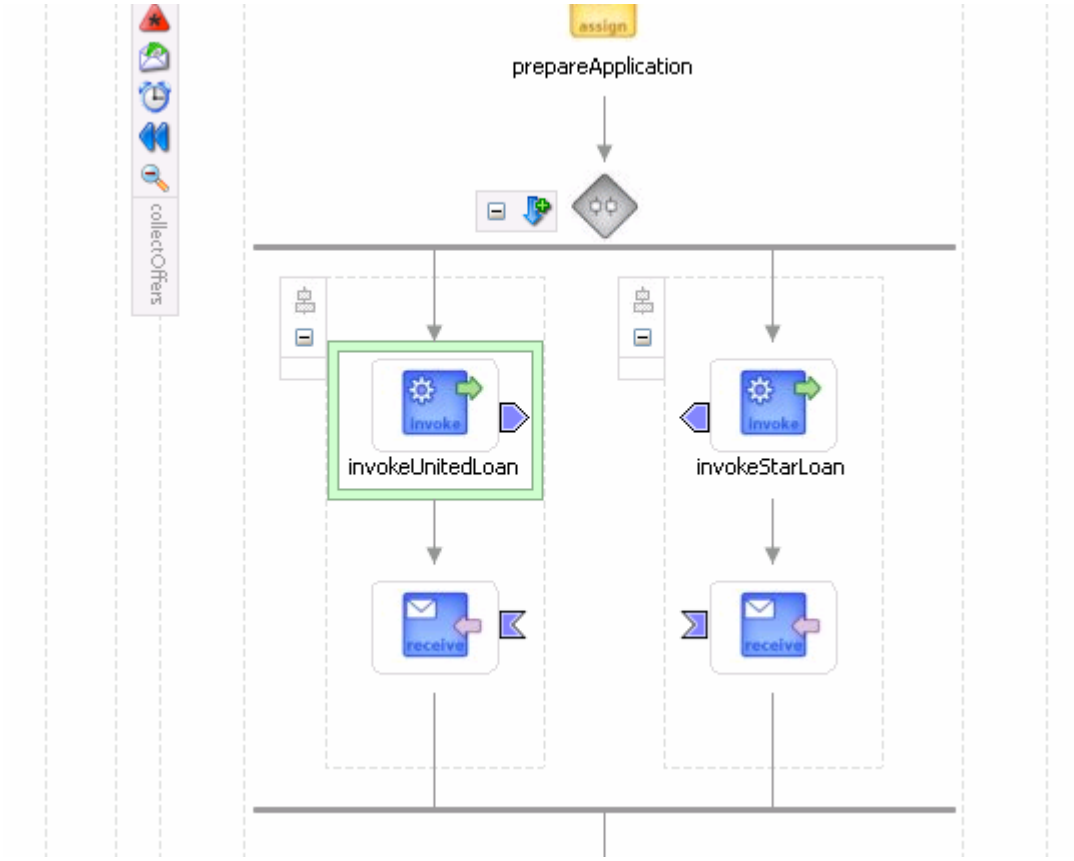
For more information about exception management, reliability and crash testing, please refer to the technotes for "Building resilient BPEL processes" and "Running crash tests on the Oracle BPEL Server" at http://otn.oracle.com/bpel.

INTERACT WITH AN ASYNCHRONOUS LOAN PROCESSOR SERVICE

Once the loan application document has been filled in, it will be sent to the two asynchronous loan processor services. In this section, we examine what is required to interact with an asynchronous service in a BPEL process. As mentioned previously, BPEL has support for asynchronous activities at its core and the Oracle BPEL Process Manager supports a "dehydration" capability so that flows are reliably and efficiently persisted to a datastore, along with all their current state information, whenever they are waiting for asynchronous events.

In addition, BPEL enables support for several standard methods of correlating asynchronous messages so that asynchronous callbacks can find their way back to the appropriate waiting process instance. Specifically, the current release of the Oracle BPEL Server supports both the correlation set mechanism, which uses message content for correlation, and the WS-Addressing specification, which uses SOAP message headers for correlation of asynchronous messages.

To see how the LoanFlowPlus process invokes the UnitedLoan service and waits for its asynchronous callback, expand the "collectOffers" scope and double-click the invoke activity in the left hand sequence labeled "invokeUnitedLoan", as shown below.



In the Invoke window you can see that this activity passes the loan application document as an input variable to the UnitedLoan service. This initiate operation will return immediately, but the next activity, the <receive> for the onResult callback, will wait until the service has called back with a loan offer. WS-Addressing is used for message correlation (by default) and is handled completely transparently by the BPEL Server. (Note that if you want to see the WS-Addressing correlation information, you can use a TCP tunnel to see the SOAP messages exchanged with the services, per the tech note on "TCP Tunneling the Oracle BPEL Server" at http://otn.oracle.com/bpel).

The BPEL source code for the invoke and receive activities which get the offer back from UnitedLoan can be seen by using the BPEL Source view or by right-clicking on the sequence icon for the UnitedLoan sequence and selecting "View BPEL Source". This source code is shown below:

```
<sequence>
    <!--  initiate the remote service -->
    <invoke name="invokeUnitedLoan"
            partnerLink="UnitedLoanService"
            portType="services:LoanService"
            operation="initiate"
            inputVariable="loanApplication"/>

    <!--  receive the result of the remote service -->
    <receive partnerLink="UnitedLoanService"
             portType="services:LoanServiceCallback"
             operation="onResult"
```

```
            variable="loanOffer1"/>
</sequence>
```

PARALLEL PROCESSING

As mentioned in the requirements section, LoanFlowPlus needs to invoke the two loan service providers in parallel since they can take an arbitrary amount of time to complete. This can be done in BPEL using the <flow> activity, which allows several actions to be taken in parallel. An example of this can be seen in the LoanFlowPlus collectOffers flow shown above which contains two parallel activities – the sequence which invokes the UnitedLoan service and the sequence which invokes StarLoan.

The BPEL source which implements this flow activity is fairly simple and is shown below:

```
<flow name="collectOffers">
  <!-- invoke first loan provider -->
  <sequence>
     <!--  initiate the remote service -->
     <invoke partnerLink="UnitedLoanService" .../>

     <!--  receive the result of the remote service -->
     <receive partnerLink="UnitedLoanService" .../>
  </sequence>

  <!-- invoke the second loan provider (in parallel with the first) -->
  <sequence>
     <!--  initiate the remote service -->
     <invoke partnerLink="StarLoanService" .../>

     <!--  receive the result of the remote service -->
     <receive partnerLink="StarLoanService" .../>
  </sequence>
</flow>
```

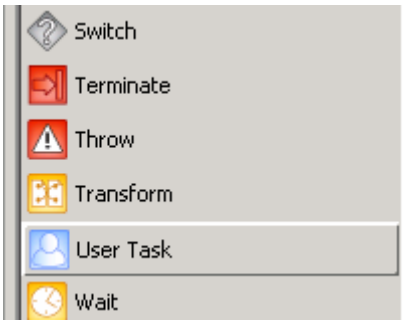
USER TASKS

BPEL has fundamental support for asynchronous services, which are easily used as a platform to integrate people and manual tasks into BPEL processes. To do this Oracle bundles a set of pre-built services with the BPEL Server that allows you to easily include human workflow into your processes. By implementing this as a true BPEL service, the interface to these services is described with WSDL and people can be included in 100% standard BPEL processes - to the BPEL process, the person/manual task looks like any other asynchronous Web service.

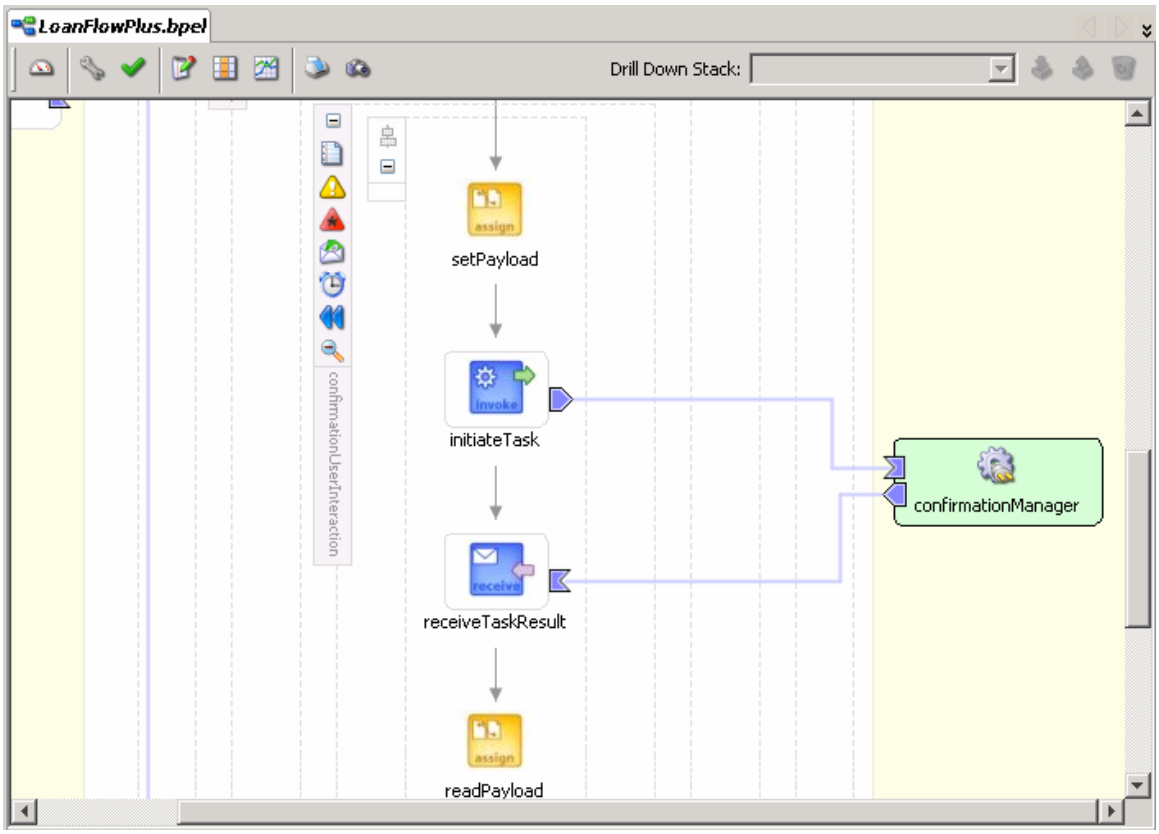
The LoanFlowPlus application illustrates manual task support in several places: for exception management, as described previously, and for the approval step where the flow waits for the customer to confirm the selected loan offer before completing.

To see how a human workflow is implemented, look in the BPEL Component Palette, it contains a User Task activity that can be dropped into a process flow. This will instantiate

a wizard that leads you through selecting assignees, approval patterns and notification channels.

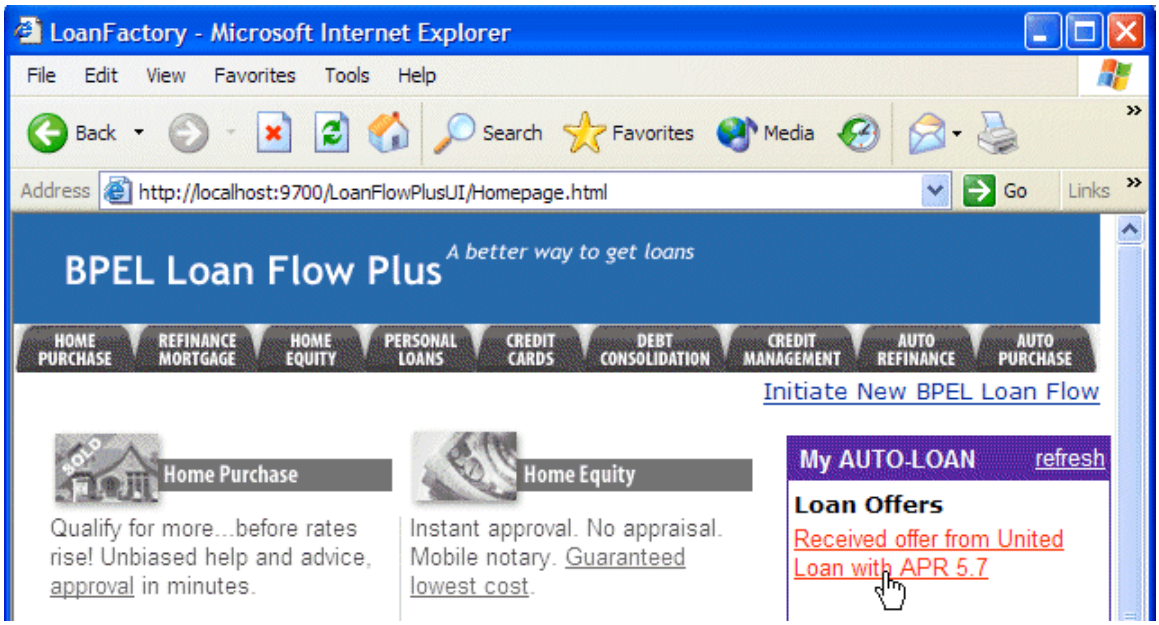


The User Task palette element is a templatized activity that allows the developer to treat a user task as a higher-level abstraction, although it is implemented as a series of BPEL activities. As you can see if you expand both the “confirmationManager” scope and the “confirmationUserInteraction” sub-scope, there is first a number of assign activities, then the process makes a call out to the “confirmationManager” partnerlink. This is the representation of the pre-built “taskManager” service within Oracle BPEL Process Manager.



APIs are provided so that clients can query the tasks assigned to a user, update the task data and complete or cancel the task. In addition, check out of tasks for group/role support expiration of tasks, escalation and other common user task requirements are implemented by the task service itself.

In the LoanFlowPlus application all user interaction is handled by custom Java Server Pages utilizing the Java API. During the customer confirmation you interact with the process through the LoanFlowPlusUI interface, and you interact with the exception handler through the ExceptionDashboardUI interface (http://localhost:9700/LoanFlowPlusUI and http://localhost:9700/ExceptionDashboardUI respectively). Oracle also supplies an out-of-the-box Worklist application that can be used (and customized) to handle human interaction within a workflow-enabled process.

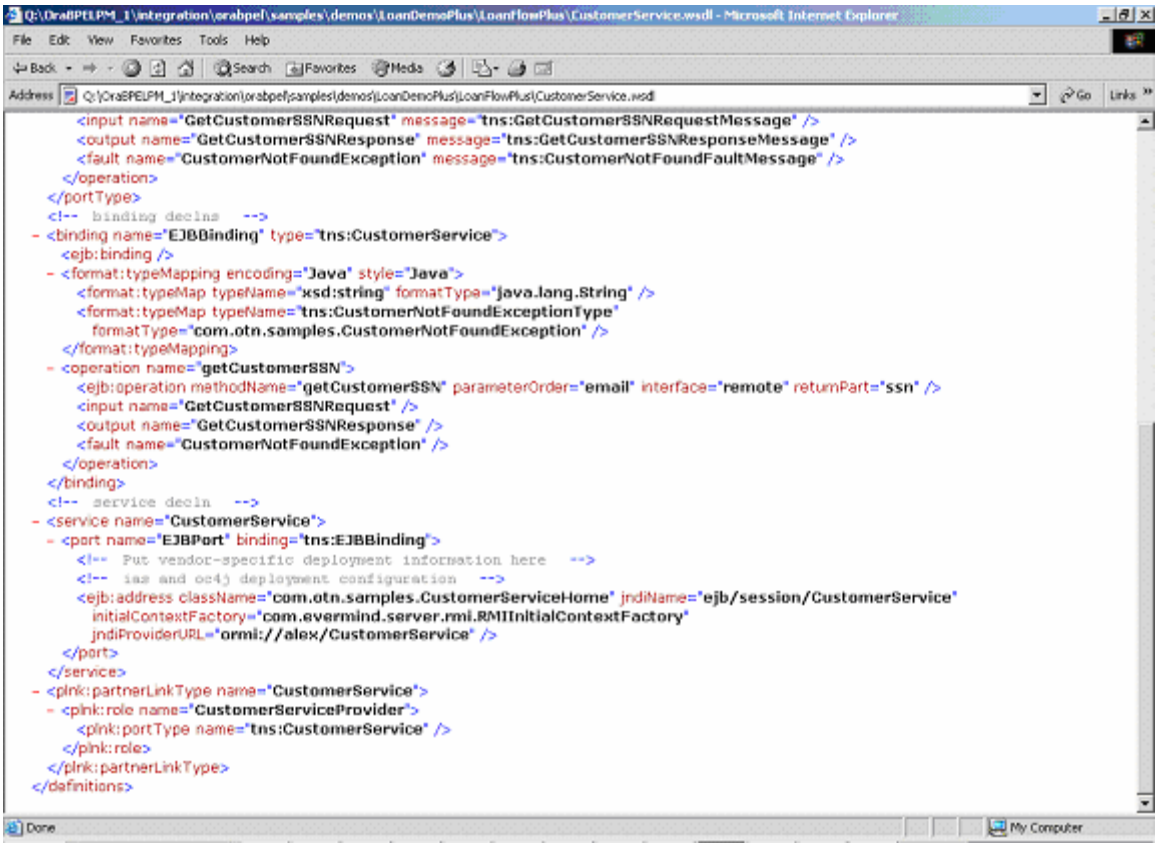


For more information on human workflow within BPEL and building task service clients, see the Workflow Services chapter in the Oracle BPEL Process Manager Developer's Guide.

WEB SERVICES INVOCATION FRAMEWORK

Frequently an enterprise has existing Java code or Java APIs for accessing back-end systems, which it wants to leverage in a BPEL process. There are 3 general approaches for this: wrapping the Java code as a Web service; "inlining" the Java code into a BPEL process; and using the WSIF invocation framework to call the Java code as if it were a Web service. The first two approaches are fairly straightforward and illustrated by several code examples shipped with the Oracle BPEL Process Manager.

As an example of the third approach, the LoanFlowPlus process uses the Web Services Invocation Framework (WSIF) to invoke an EJB which returns a social security number for the customer. From your BPEL process you can treat the EJB "service" just as you do any other service which has a WSDL interface and create a partner link for it in your BPEL process. Functionality offered by the Enterprise Java Bean is mapped directly to the services WSDL interface through an EJB binding. You can see this binding in the WSDL shown below:



This means that the EJB is being accessed as a WSDL described service, where WSIF is mapping the operations to EJB methods.

Note that JDeveloper 10.1.3 will automatically generate WSDLs with Java and EJB bindings for POJOs ("plain old Java objects") and EJBs, in addition to the currently supported SOAP bindings.

BUILD A CUSTOM USER INTERFACE FOR INITIATING THE BPEL PROCESS

All BPEL processes are themselves Web services. In addition, the Oracle BPEL Process Manager also provides a Java API for invoking deployed BPEL flows, fetching state/status information from active instances, etc. Frequently a BPEL process will be instantiated from a portal or other custom user interface – for example, the LoanFlowPlus process has a JSP loan interface that customers can use to initiate new loan applications, see the offers they have received and approve offers. The LoanFlowPlus UI is deployed onto the same application server as the Oracle BPEL Process Manager and full source for it can be found in your samples at:

Oracle_home\integration\orabpel\samples\demos\LoanDemoPlus\LoanFlowPlusUI

JavaDocs for the Java API for initiating a deployed BPEL process can be found in:

Oracle_home\integration\orabpel\docs\apidocs\index.html

In addition, a JSP tag library is available to make it easy to use this Java API from a JSP (this tag library is used by the LoanFlowPlusUI sample) and developers can also use the

Web services SOAP/WSDL API to invoke BPEL processes. The Web services approach allows BPEL flows to be invoked and accessed from any language or toolkit that supports Web services.

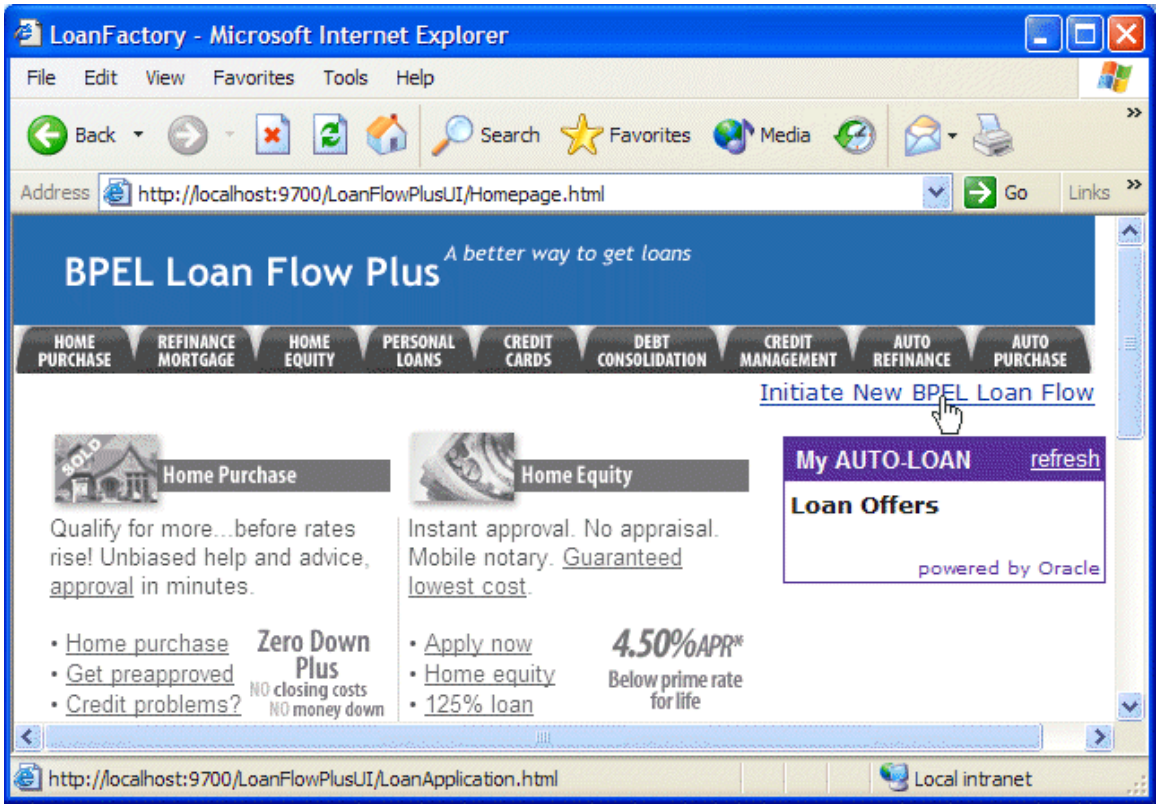
INITIATE THE LOANFLOWPLUS PROCESS

In this section you will use the portal UI to initiate and complete a LoanFlowPlus instance and the BPEL Console to view the status, audit trail, debugging information, performance metrics and other information gathered automatically by the Oracle BPEL Server.

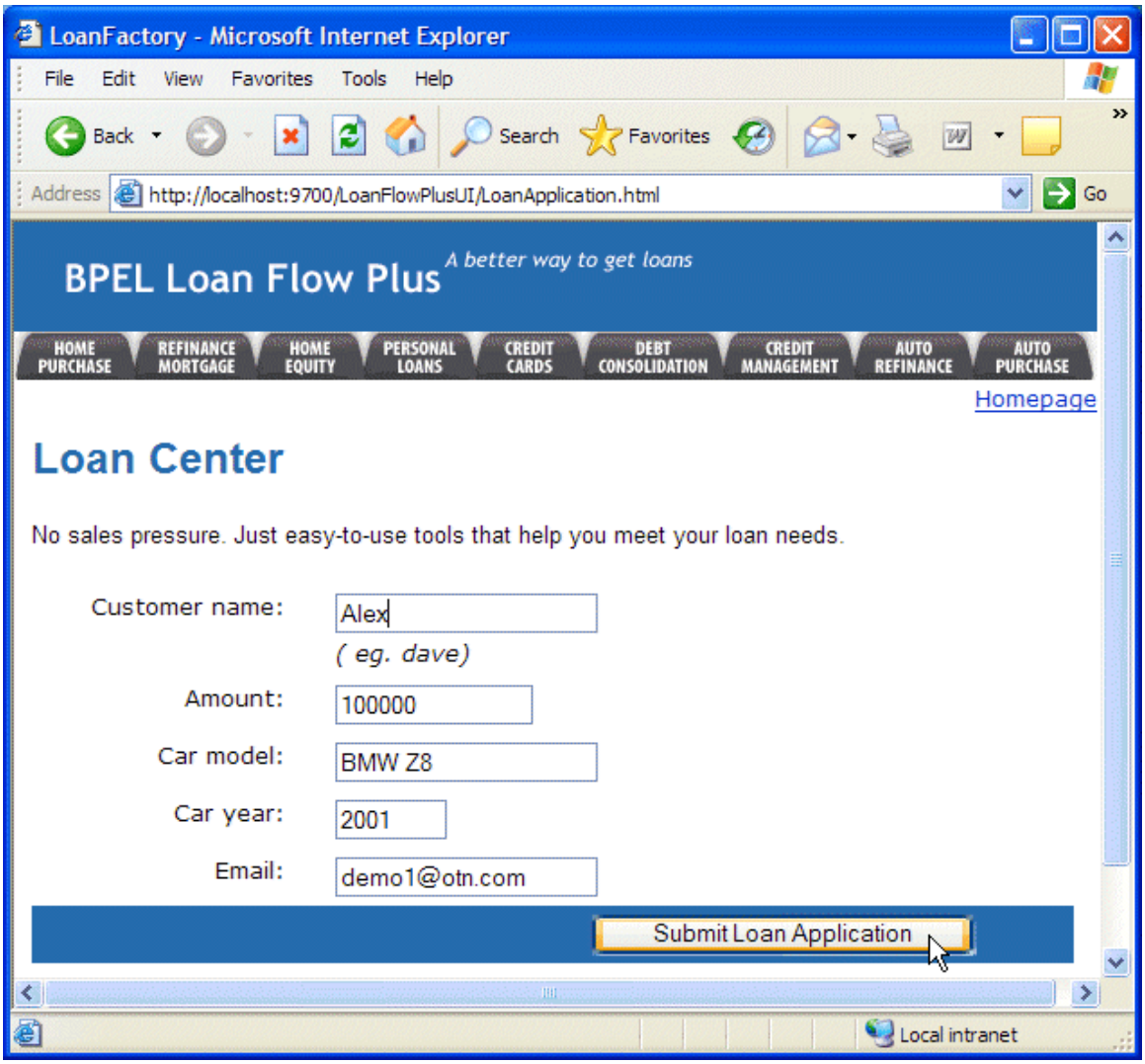
1 The LoanFlowPlus BPEL process was compiled and deployed, along with its dependent services and the portal UIs, when you executed the obant command at the beginning of this section. You could of course also build and deploy it from the Designer, as you did in the previous section. You should now point a browser at the portal UI located at:

http://localhost:9700/LoanFlowPlusUI/Homepage.html

2 Select the "Initiate New BPEL Loan Flow" link on the portal page, as shown below:



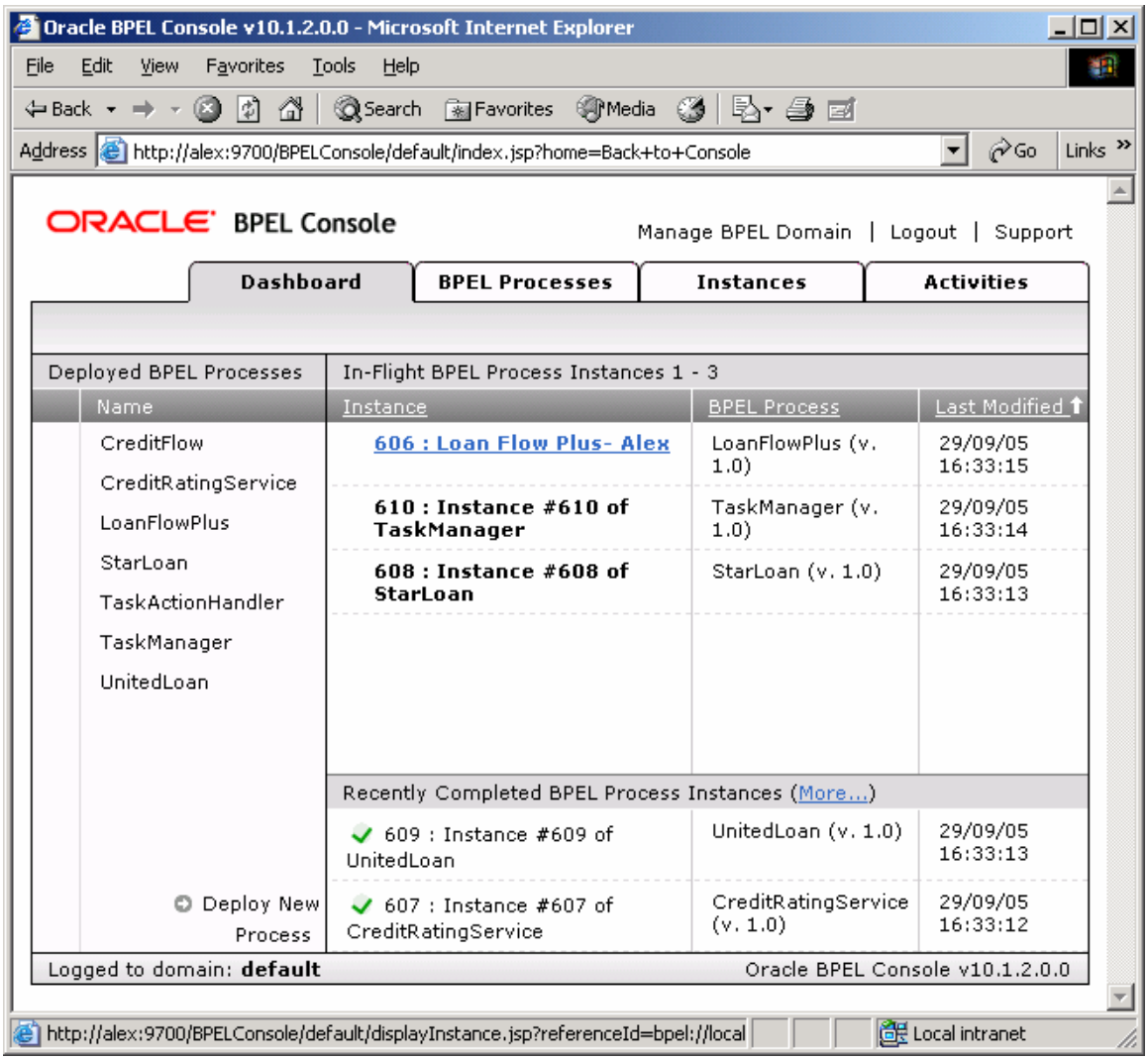
3 You will now see a web page which allows you to submit a loan application to initiate a new LoanFlowPlus instance. You can change the fields in the UI, if you choose, and click the Submit Loan Application button to initiate the new process instance:



REVIEW THE VISUAL AUDIT TRAIL

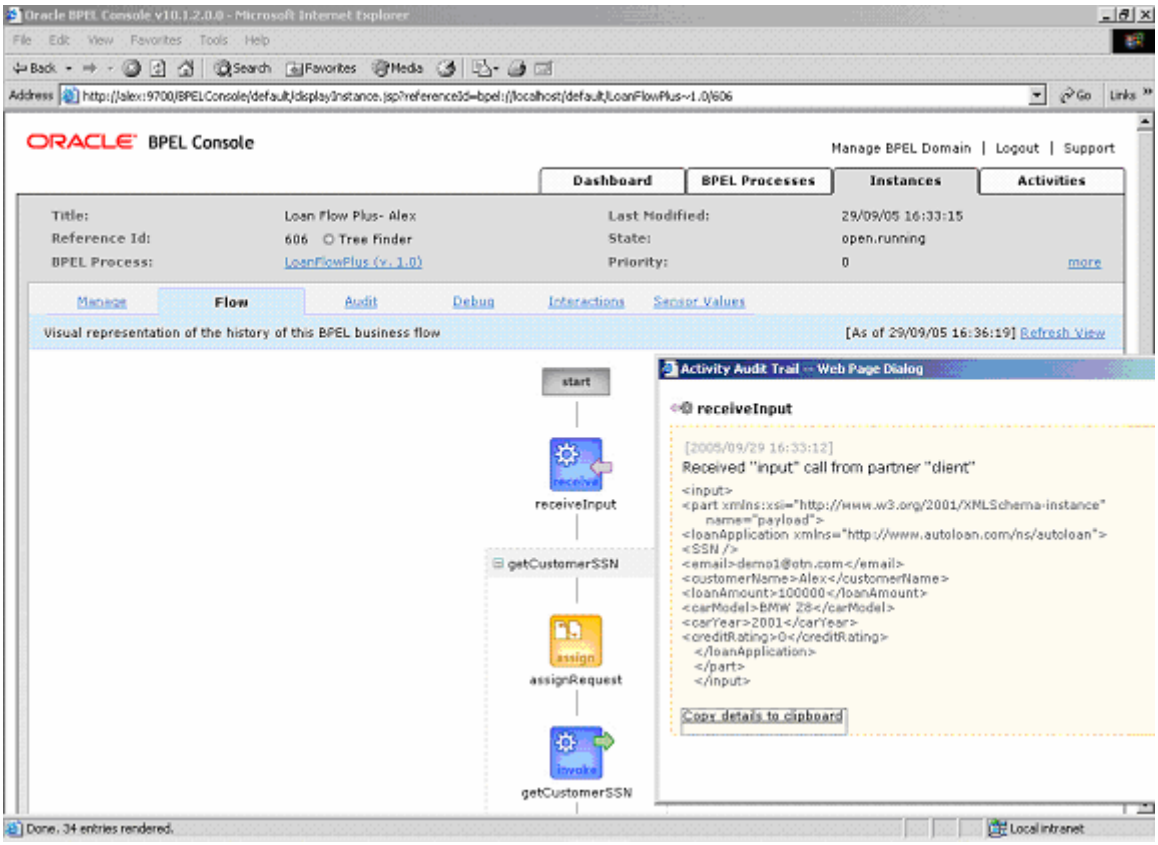
Next you will put on the hat of a developer or administrator and use the BPEL Console to view the audit trail and other status information regarding this process.

4 To do this, point a different browser window at:
http://localhost:9700/BPELConsole/
and log in if necessary (the default password is “bpel”). You should see several in-flight instances and several completed instances of BPEL processes. Select the active instance of the LoanFlowPlus application as shown below:

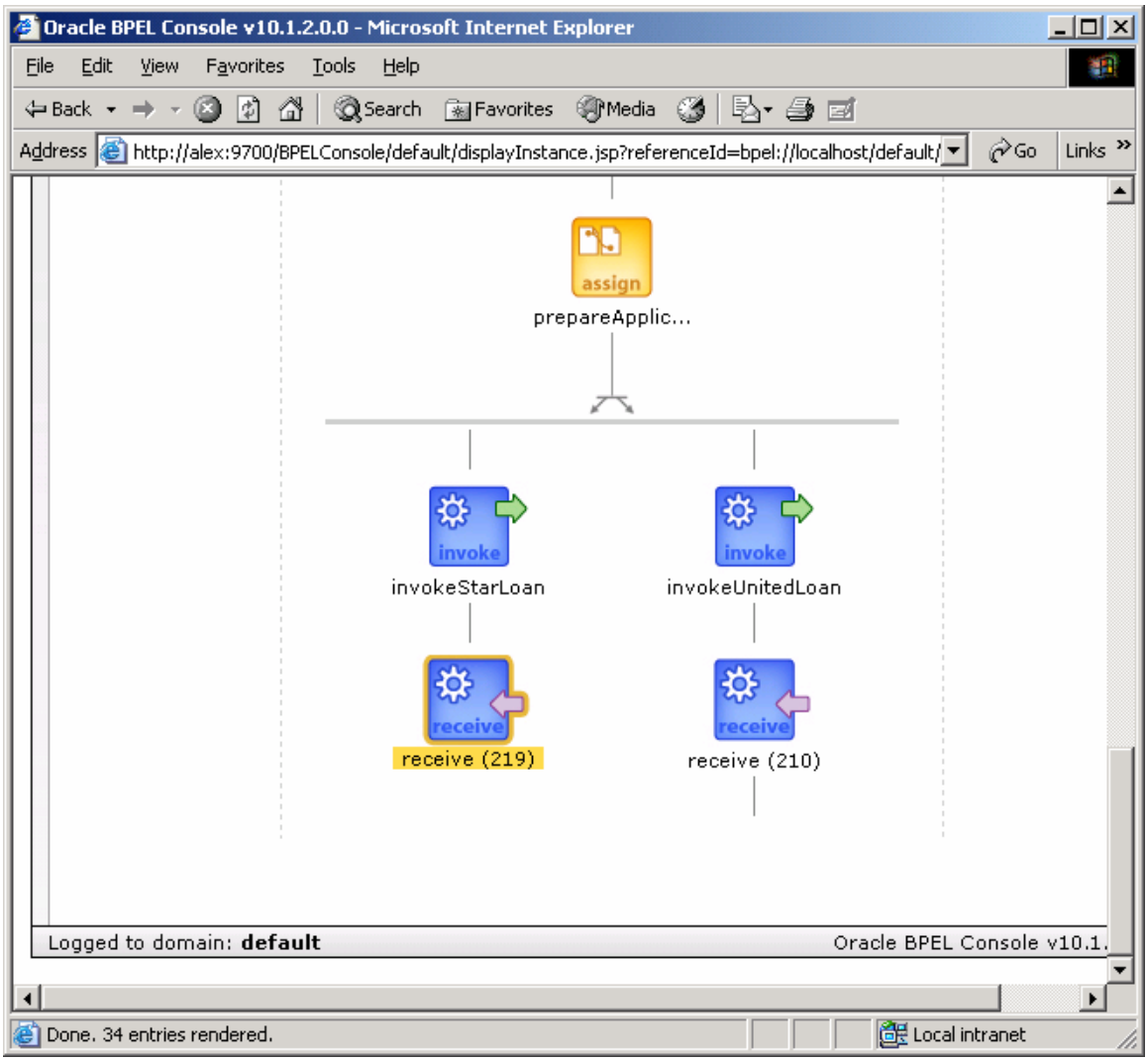


Note that everything that you see in the BPEL Console is available via an API as well. In fact, the BPEL Console itself is just a set of JSPs that can serve as a code example of how to use the BPEL Server APIs. Developers frequently use these APIs to build custom process dashboards, search screens and other interfaces so users and administrators are able to more efficiently access and manage process instances. This is particularly common as applications go into production where there may be thousands or even millions of process instances to manage.

5 Once you select the LoanFlowPlus instance in the console, you will see the “visual audit trail” which shows the current status of the process and its execution history. This audit trail is automatically generated and maintained by the BPEL Server, though developers and administrators can control what information gets logged. If you click on an activity in the visual audit trail, you will see detail information for that activity. For example, in the picture below, the client initiation has been selected and so you see the XML input message which was sent to kick off the flow.



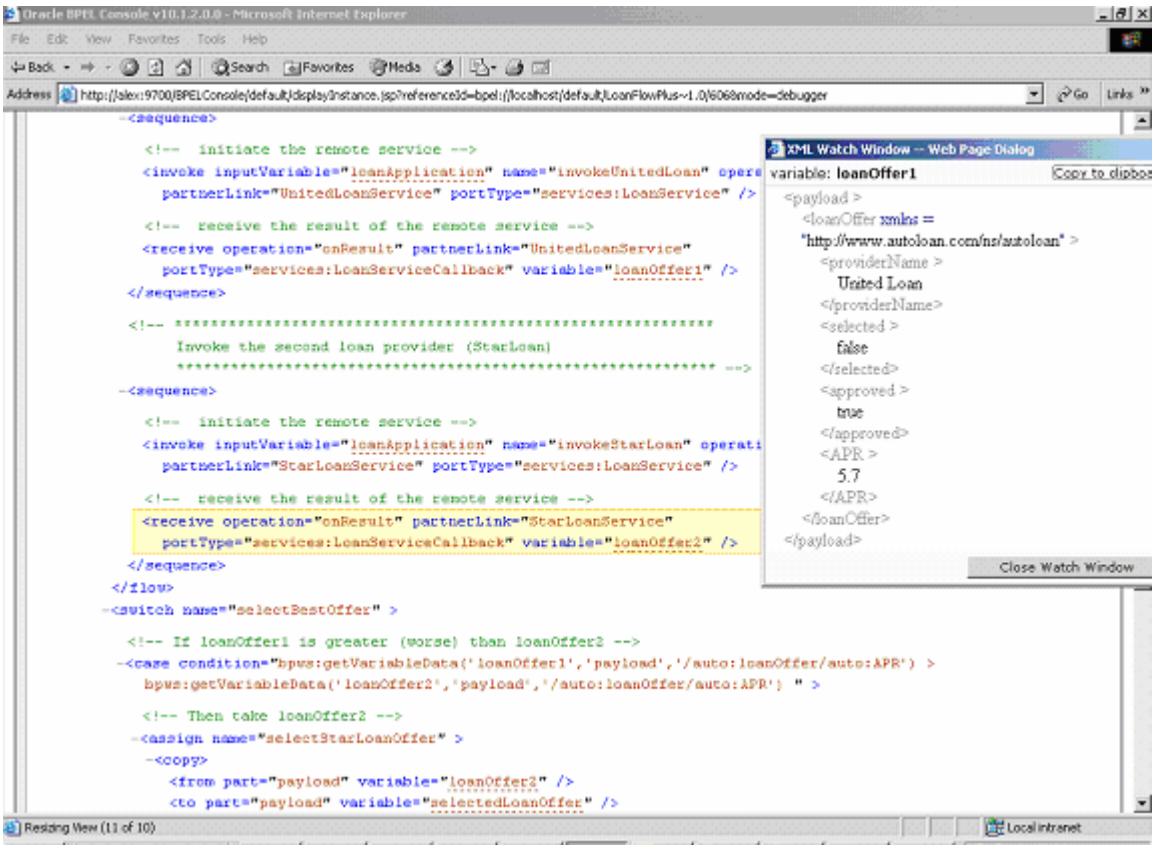
6 If you scroll down to the bottom of the visual audit trail, you will see the latest status for this instance, in this case that the StarLoan service has been initiated but the flow is waiting to receive a response (as indicated by the orange highlighting), however the UnitedLoan service has already called back with a loan offer.



7 You can also select the “Audit” tab, below the instance details at the top of the page, to see a text-based audit trail. The text audit trail includes such information as the messages exchanged with services and timestamps for when each activity was started or completed and again is both configurable and accessible via API.

DEBUGGING THE IN-FLIGHT BPEL PROCESS

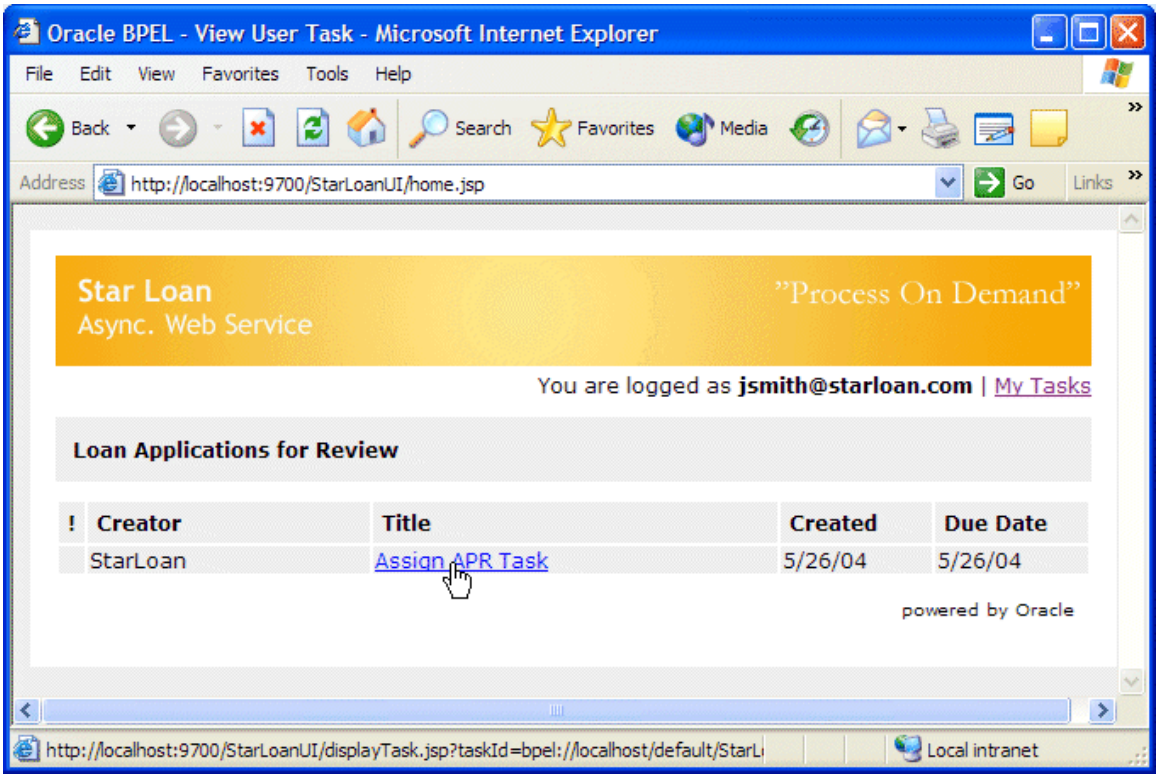
8 If you now select the “Debug” tab, you will see the BPEL Debugger which takes the BPEL source that implements this process and matches it up against the state of this particular instance. Points in the code where execution is currently paused are highlighted and as you can see below, the process is currently waiting for the StarLoan service to call back with a loan offer.



9 You can select variables in the debugger to inspect their current values. Shown above is the value of the loan offer variable that contains the result of the UnitedLoan service callback operation. As you can see, UnitedLoan has approved the loan application and returned a 5.7% interest rate loan offer.

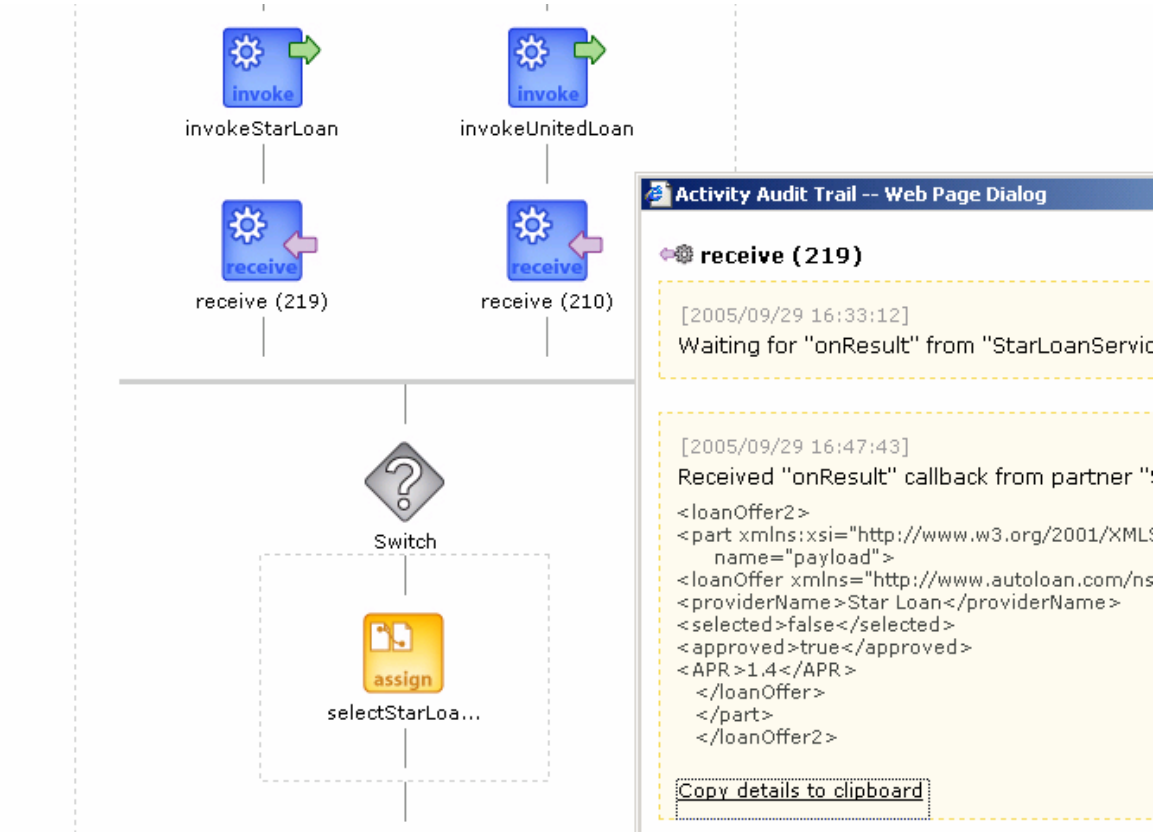
COMPLETING THE ASYNCHRONOUS STARLOAN SERVICE

10 If you now want to complete the process, you can bring up a StarLoan customer service rep dashboard, since StarLoan requires manual processing of loan applications. Naturally, StarLoan has also been implemented as a BPEL process (though implementations are available where StarLoan is built with other Web services toolkits, such as Microsoft .Net, Apache Axis, etc) and serves as an additional BPEL code example. To play the role of the StarLoan loan officer, point a browser at: http://localhost:9700/StarLoanUI/home.jsp



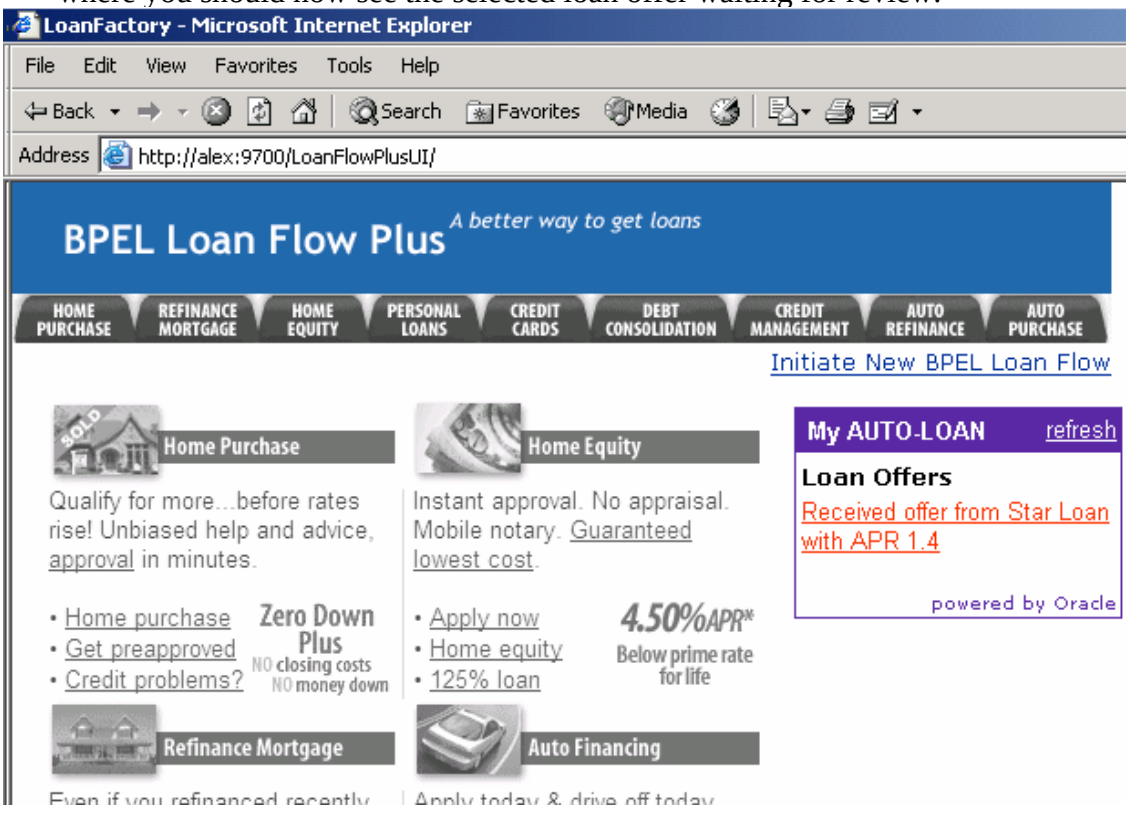
Here you should see at least one pending loan application, which you should select, enter an interest rate (go ahead and make it a nice low one as long as you are approving your own loan application...) and click the approve button. This will cause the StarLoan service to return a callback to the LoanFlowPlus process.

11 If you refresh its audit trail, you should see that it has moved along in its processing, having received the StarLoan callback with the second loan offer, selected the best loan offer and is now waiting for the customer to approve the offer.



COMPLETING A USER TASK

12 As the final step in the process, you could refresh the customer loan portal homepage where you should now see the selected loan offer waiting for review.

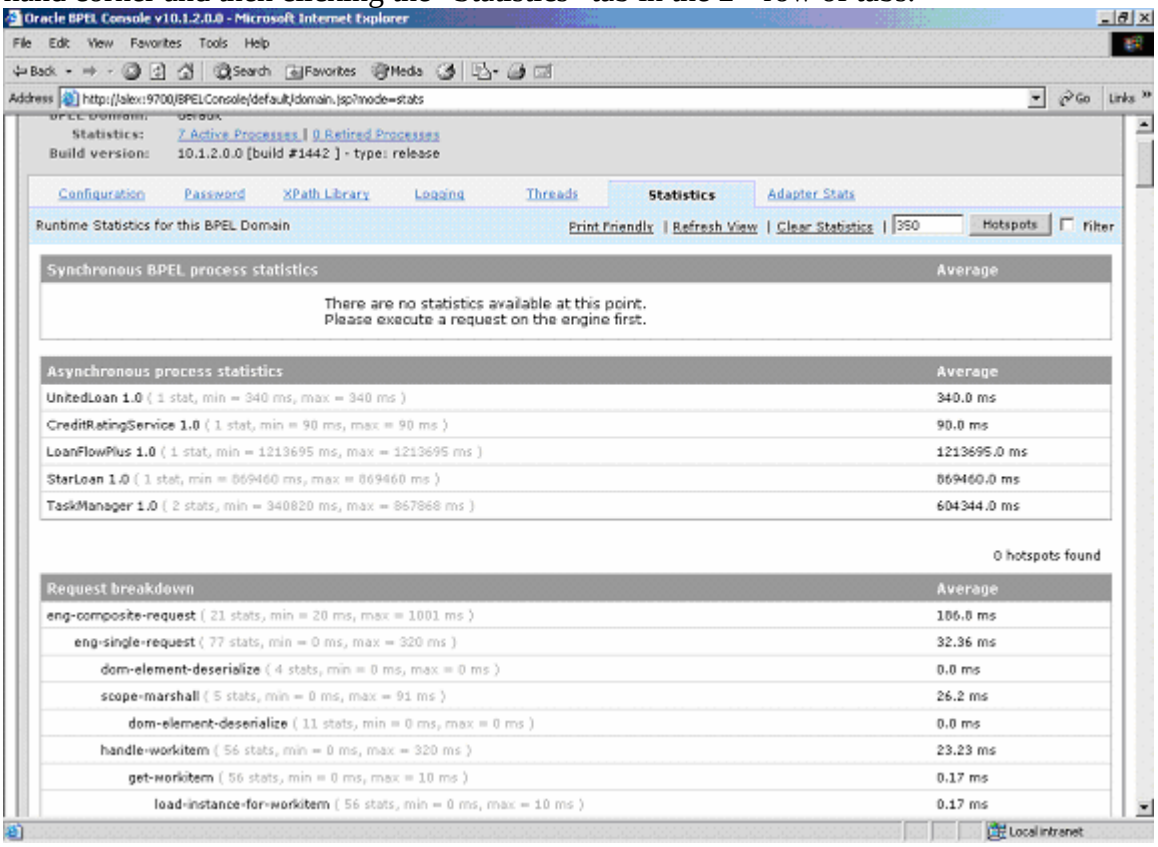


If you select this loan offer you would get a page that allows the customer to accept this offer. This will complete the execution of this instance of the LoanFlowPlus BPEL process.

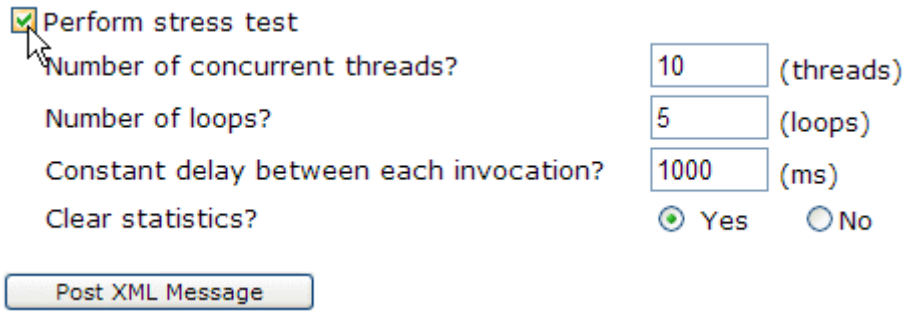
PERFORMANCE TUNING

Frequently performance (including throughput and response time requirements) is a critical requirement for effectively implementing a BPEL process. Built-in to the BPEL Server and Console is a performance-tuning framework that assists developers to understand the performance of their applications and identify and resolve bottlenecks (whether in the developer's code, an external service or the BPEL Server itself).

13 You can see a page with the performance data for your completed flows by going to the BPEL Console, selecting the "Manage BPEL Domain" link in the upper right hand corner and then clicking the "Statistics" tab in the 2nd row of tabs.



As you have seen, the BPEL Console allows you to test any of your deployed processes through an automatically generated HTML form interface or by passing specific XML message content to it. This test page includes stress test capability that makes it easy to do load testing of a deployed BPEL process and then view the performance statistics of the flow under stress.

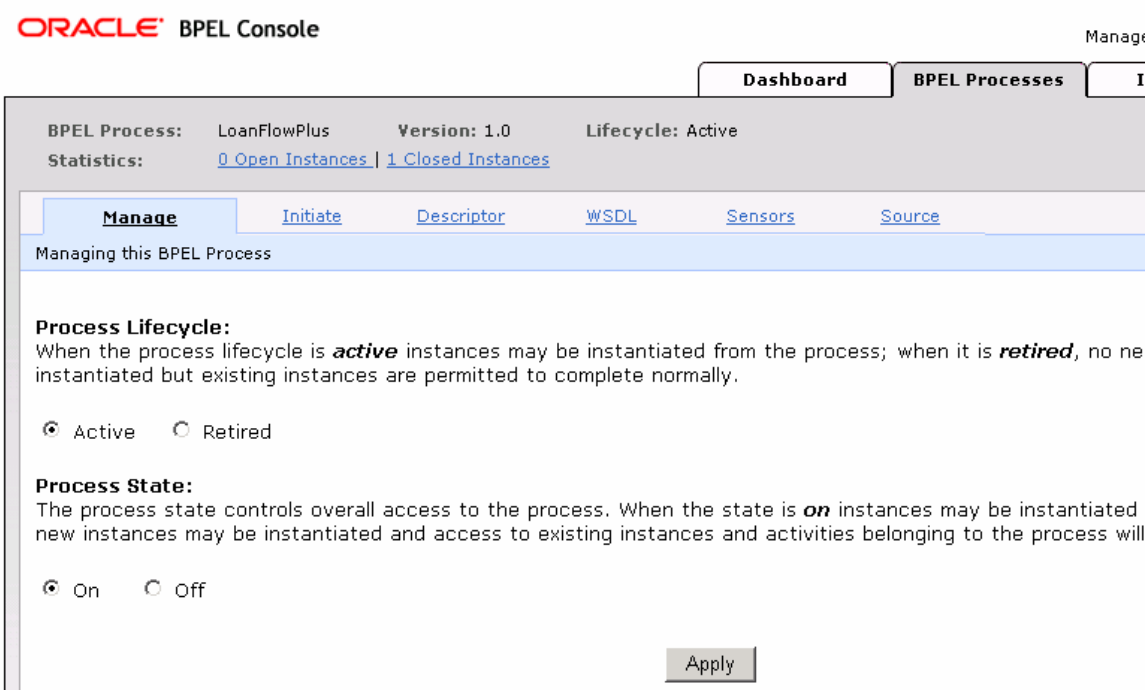


Note: please don't try the built-in stress test capability with the OracleLite database bundled with the developer installation of Oracle BPEL PM!

PROCESS LIFECYCLE MANAGEMENT

The Oracle BPEL Process Manager and BPEL Console also include support for process lifecycle management. For example, side-by-side versioning is supported so that new implementations of a process can be "hot deployed" without disturbing current in-flight instances of prior implementations of the process. In addition, the BPEL Console can be used to turn off an entire process, undeploy a process, or "retire" a process such that existing instances can complete, but new instances of the process will not be allowed.

14 This functionality is available through the process management tab. To see this tab, select the name of a deployed BPEL process in the BPEL Console and then click the "Manage" tab on the left.

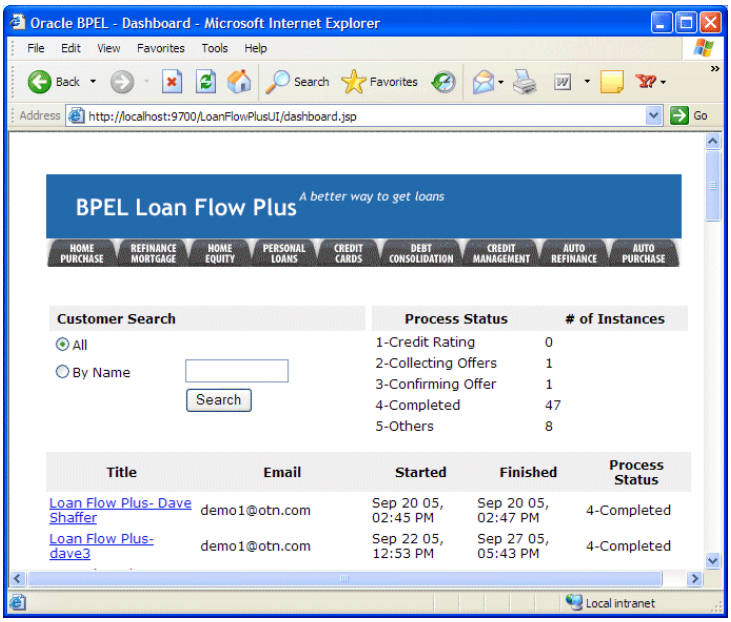


BUILD A REPORTING DASHBOARD

A “killer feature” of automating a business process is often to provide greater visibility into process state information. This is often realized by building a process “dashboard” which displays aggregate summary information or process instance state information. For example, an executive may want to know how many loans are currently at each stage of processing, the average time to present a loan offer to a customer, or other key performance indicators against live and completed process instances.

The Oracle BPEL Process Manager maintains a great deal of information regarding process state and, as mentioned, makes this information available via API. In addition, several simple reporting dashboard templates are shipped with the BPEL Server or are available from professional services and support organizations. Such a template is included with the LoanFlowPlus process and can be accessed via the URL:

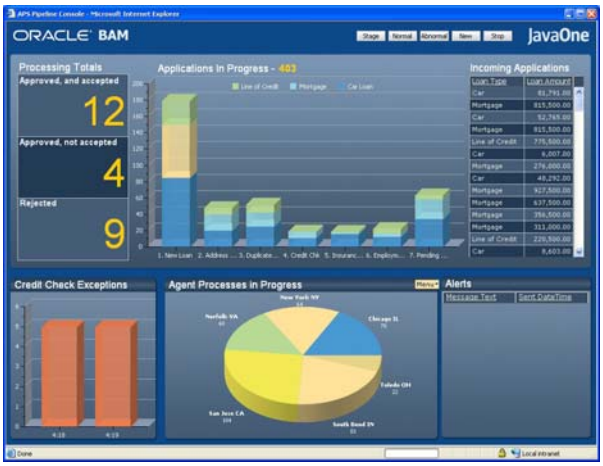
http://localhost:9700/LoanFlowPlusUI/dashboard.jsp



In addition, the 10.1.2.0.2 release of Oracle BPEL Process Manager adds some built-in reports to the BPEL Console to enable process optimization. An example is the process time distribution report shown below.

Activity Name	Activity Type	Count	Average Time(seconds)	Average Execution Time Details
invokeCR	invoke	125	1.87	
invokeUnitedLoan	invoke	69	0.36	
receiveInput	receive	116	1.31	
selectBestOffer	switch	85	0.32	
prepareTitle	assign	93	1.0	
initiateTask	invoke	45	0.25	
receiveTaskResult	receive	75	0.65	
replyOutput	invoke	92	0.15	
assignRequest	assign	128	1.5	
copyRating	assign	45	1.2	

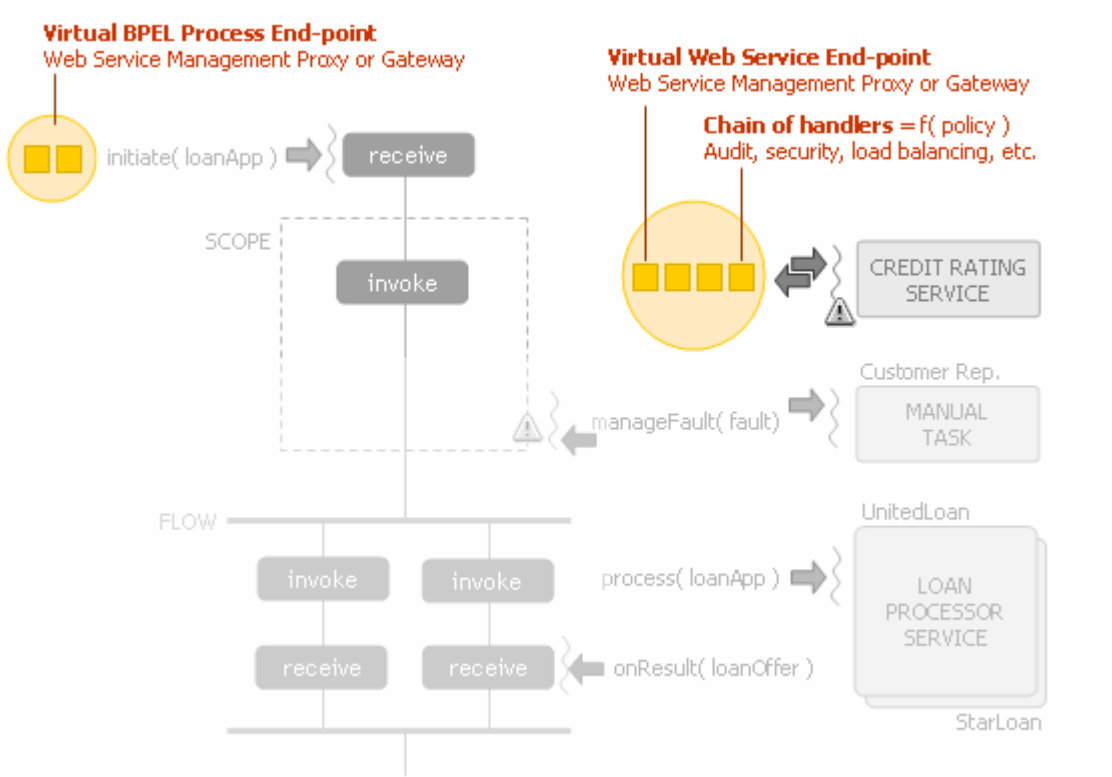
Oracle also provides a Business Activity Monitoring (BAM) product that integrates tightly with the Oracle BPEL Process Manager. Oracle BAM gives business executives the ability to monitor their business events and processes in the enterprise, to correlate messages and define KPIs (key performance indicators) and most importantly to define alerts and take corrective action based on real-time information accessed through a live-data, thin-client web interface.



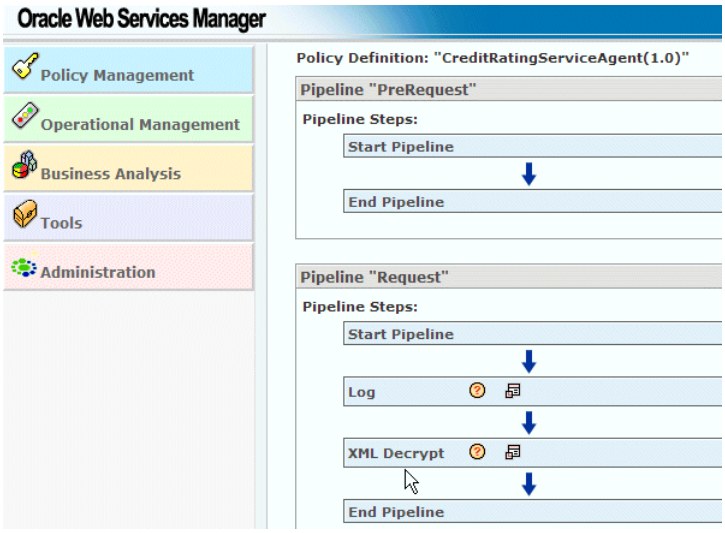
Oracle BPEL Process Manager integrates easily with Oracle BAM. For example, the BPEL Designer includes support for "sensors" which make it easy to push events from BPEL processes out to the BAM server. However, the BAM server can gather events from any source within the enterprise (JMS messages, DB events, etc) that can write to a JMS queue and is not limited to BPEL process events.

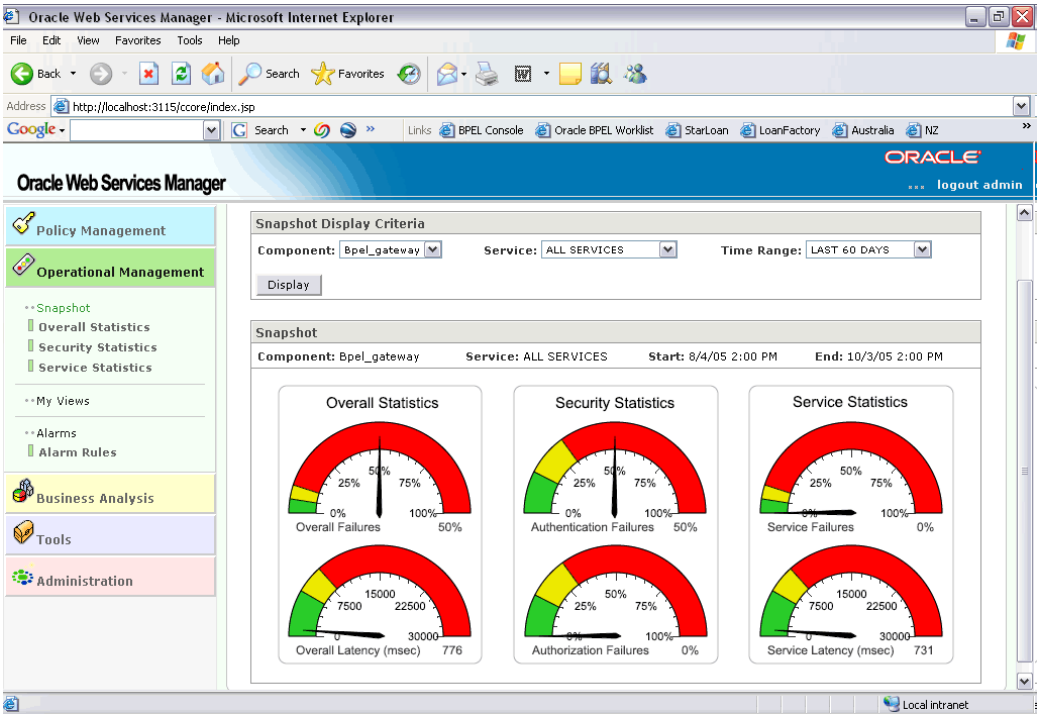
INTEGRATION WITH A WEB SERVICE MANAGEMENT SOLUTION

The question frequently comes up as to secure and manage both services and the BPEL processes that orchestrate them within an enterprise adopting SOA. A WSM solution, like the Oracle Web Services Manager (OWSM), provides features such as authentication, authorization, encryption and logging for services and processes through the definition of policies. As shown in the diagram below, Web services management solutions typically provide proxies or gateways in front of Web services. This includes both "back-end" Web services which may be implemented in many different technologies as well as BPEL processes deployed to the Oracle BPEL Server, or other BPEL servers, since a BPEL process is itself a Web service.



Here, OWSM will provide a "managed" end-point, which looks to the BPEL process just like the underlying Web service. The BPEL process does not even need to know that it is connecting to a managed end-point rather than the service itself. OWSM (or other products which support the same standards) then provides a pipeline of steps so that encryption, authentication, service virtualization and other key requirements can be implemented in a uniform manner. Likewise, OWSM can be placed logically in front of Oracle BPEL PM to manage the connections to BPEL processes themselves.





For more information on OWSM, and its use with the Oracle BPEL Process Manager, see http://otn.oracle.com/products/webservices_manager.

90+ Additional Examples

We believe that "samples are a developer's best friends" and therefore try to continuously increase the number of sample BPEL projects which ship with the BPEL Server.

Here is a quick overview of how those samples are organized:

ORACLE_HOME*\integration\orabpel\samples\demos

Contains a set of demonstration BPEL processes, some examples are listed below:

AmazonFlow showcases how an Amazon Web service can be integrated in a BPEL process.

BankTransferDemo showcases how to incorporate XA transactions in BPEL processes. BankTransferFlow orchestrates EJBs in an atomic transactional fashion by using the EJB WSIF binding so they can participate in the BPEL engine's own transaction. BankTransferFlowWithCompensation demonstrates how XA and compensating transactions can be mixed and matched in a BPEL process.

CheckoutDemo showcases how to do a 2-step receive…reply…receive…reply between the client and the BPEL process. This is sometimes called "UI orchestration" and is not always a best practice, but can be useful sometimes.

GoogleDemo showcases how a Google Web service can be integrated into a BPEL process.

HotwireDemo showcases how to use correlation sets to create sophisticated and stateful interactions between a client and a BPEL process.

IBMSamples showcases how the BPEL samples shipping with BPWS4J can be executed on the Oracle BPEL Server.

LoanDemo showcases how to integrate a synchronous credit rating service and two asynchronous loan processor services into an end-to-end loan procurement application with a JSP UI to initiate the process and view loan offer results.

LoanDemoPlus is an extension to the LoanDemo sample showcasing Java embedding, exception management, including manual processing steps, and development of a richer custom user interface.

ParallelSearch showcases the ability of the BPEL server to perform parallel synchronous invocations.

PriorityDemo showcases the support in the BPEL server for setting priorities for different processes and instances and how those affect execution.

ResilientDemo showcases how a BPEL process can be instrumented to properly manage run-time exceptions (see http://otn.oracle.com/bpel for more information).

SalesforceFlow showcases how the Salesforce.com sForce web services can be integrated into a BPEL process (including authentication, session management and https services).

TimeOffRequestDemo showcases how to model user interactions and workflow tasks within a BPEL process

ORACLE_HOME*\integration\orabpel\samples\hw

Contains the source code for the tasklist/worklist application which ships with the product providing a GUI for viewing and acting on tasks assigned from BPEL processes (or elsewhere) using the Workflow service bundled with BPEL PM.

ORACLE_HOME*\integration\orabpel\samples\interop

Contains a set of BPEL projects showcasing the interoperability of the Oracle BPEL Process Manager with Web services implemented with Microsoft .Net, Apache Axis and BEA WebLogic.

ORACLE_HOME*\integration\orabpel\samples\references

Contains a BPEL project for each activity and concept defined in the BPEL language. A good way to learn about and try out a specific feature before integrating it into a larger BPEL process.

ORACLE_HOME*\integration\orabpel\samples\tutorials

Contains a set of fairly simple BPEL processes targeting the various tasks a BPEL developer is exposed to, including: building your first BPEL process, invoking a synchronous service, invoking an asynchronous service, manipulating data elements, defining parallel branches of execution, managing faults, managing timeouts, defining correlation sets, modeling user interactions, calling Java components, manipulating XML arrays, defining complex multi-step routing, integrating XSLT and XQuery transformations, sending and receiving emails, interactions with JMS destinations and defining custom WSIF bindings.

ORACLE_HOME*\integration\orabpel\samples\utils

Contains a set of building block services shared by the BPEL samples. Some of these may be useful samples as well. For example, the LoanFlow demo catches exceptions thrown by the CreditRating service and provides an example of how to catch and handle exceptions in BPEL. The CreditRating service is just a building block that is useful for the LoanFlow, and other, demos. However, it can also be sued as an example of how to return an exception to the client of a BPEL process.