Informatica® ActiveVOS
9.2.6

Designer

Informatica ActiveVOS Designer
9.2.6
December 2024

© Copyright Informatica LLC 1993, 2024

Table of Contents

Chapter 7: Implementing a BPMN Task or Event in BPEL. 127

Chapter 8: Implementing a BPMN Gateway or Control Flow. 182

Chapter 9: Using Variables. 204

Preface

This module contains information about how to create assets in Informatica ActiveVOS.

CHAPTER 1

Welcome to Informatica Process Developer

This chapter includes the following topics:

About Informatica Process Developer

Process Developer automates many of the requirements to create a valid BPEL process. This makes it easy to build a process.

As you build a process, by defining the participants and their activities, Process Developer generates executable BPEL code.

Here is a typical picture of a Process Developer process.
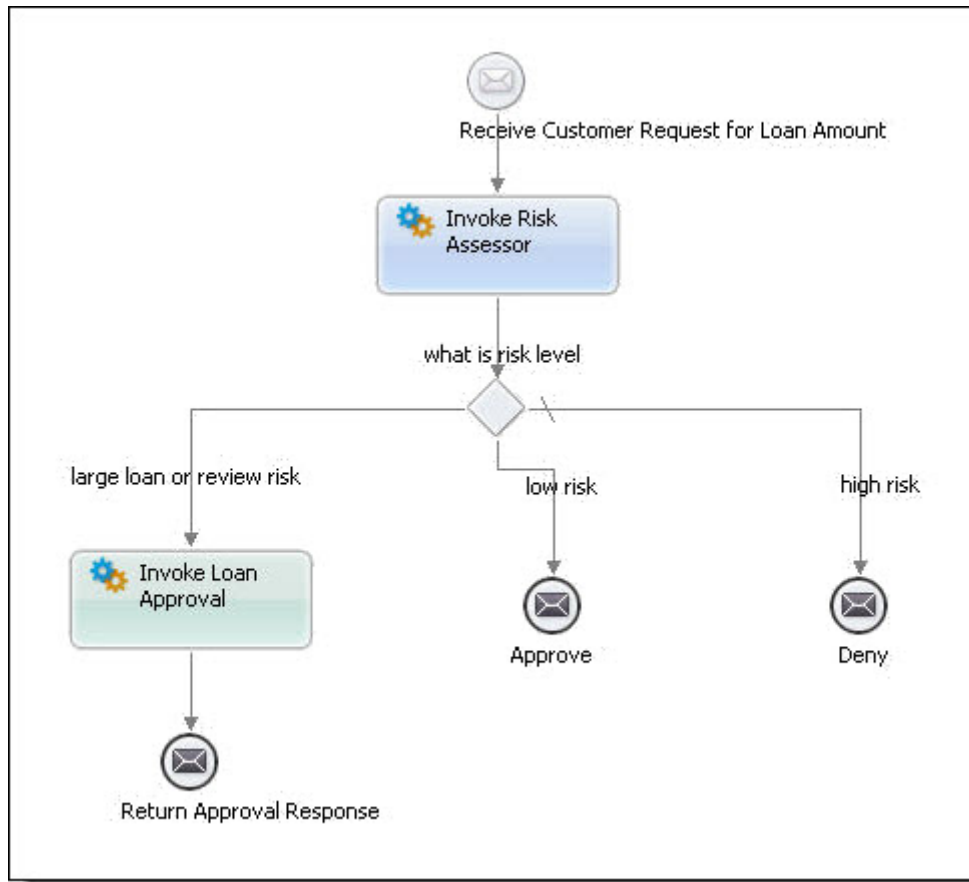
Note that this process is easy to understand for non-technical and technical people alike.

Here is a snippet of the BPEL code corresponding to the illustration above, as generated by Process Developer.

When your process is complete, you can deploy it to a server that can be started and run from Process Developer. This allows easy testing of your process before you put it into production.

Process Developer incorporates many technologies, including:

- XPath and XQuery expressions for data manipulation
- REST or Java interfaces
- BPMN diagramming

Working in the Eclipse Environment

The Process Developer is a series of plugins to the Eclipse integrated development environment. Eclipse is an open platform for tool integration built by an open community of tool providers. Process Developer takes advantage of the Eclipse Workbench features to provide BPEL building and orchestration capabilities.

You can install Process Developer in the following ways.

- Install Process Developer as a stand-alone application
  The installation archive extracts Process Developer as an Eclipse application. Start the application by launching te Process Developer executable.
- Install Process Developer plugins into your Eclipse environment.
  In the installation archive that you downloaded, use the `Process Developer_plugins.zip` (or `Process Developer_plugins.tar.gz`) to extract and then copy the required features and plugins into your Eclipse installation. In Eclipse, open the Process Developer perspective.

For installation details, see the `readme` file in the root of folder where you extracted your Process Developer installation files.

For more information about the Eclipse community, see http://www.eclipse.org.

Migrating from Earlier Versions

Required Reset for Process Developer Perspectives

If you maintain your existing Workspace when you upgrade Process Developer, we recommend the following best practices so that you can take advantage of new features:

- Reset both of the Process Developer perspectives. From the Window menu, select **Reset Perspective** for the perspective in focus.
- Delete the current server configuration and add a new server. The Process Developer embedded server runs from a plug-in that includes version details. Each time you upgrade to a new version, you must delete the old server configuration by navigating to **Window > Preferences > Server > Runtime Environments** and select Remove to delete **Process Developer Embedded Server**. Then select **Add** from this same dialog to install a new server runtime environment.

Migrating from Process Developer Versions Prior to 9.0

There are a few changes to be aware of for new versions of Process Developer.

- Existing BPEL processes have an associated visual layout file that includes all layout and annotation information (`.vbpel` files). New BPEL files contain this information, eliminating the `vbpel` file and making it easier to move BPEL files to another location without losing annotations. To delete the `vbpel` file for existing processes, open and save your older processes. This is not required. This feature does not apply to BPEL 1.1 processes or processes designed in the Classic style.

- The Process Editor palette offers a choice of BPMN-Centric (the default) or the existing BPEL-Centric stylesheet. Existing processes can be opened with either stylesheet with the exact same results. To use the BPEL-Centric palette, change the Layout Preference.

- You can add an XQuery nature to existing projects to take advantage of the XQuery editing and runtime tools. Right-mouse click on a project and select Add XQuery Nature.

- POJO (and XQuery) custom functions can be listed in the Expression Builder and deployed in your deployment contribution. There is no longer any function context set up required for new POJO custom functions.

- If you have existing XQuery functions, be sure to read the Release Notes (elsewher in this help) for possible changes you may need to make to your functions.

- A newly generated B-unit ant script contains targets and parameters for code coverage. You can manually add these ant tasks to your existing B-unit ant scripts. You can also add XQuery modules as B-unit resources.

- You can migrate running process instances to run against a new process version. As a first step, open your existing process, modify it slightly by moving an activity a bit (mark the process as dirty), and save the process without making any other changes. Update the PDD to enable the Migrate Version option. Then deploy the contribution that contains the unchanged process structure and updated PDD. After you have deployed the Version 9 contribution, you can then make structural changes to your process and migrate running process instances to the new version.

- Version 9 uses a newer version of the Saxon XSL parser that does not support the undocumented and non-standard `@value` attribute for the `<xsl:param>` element. If your older B-unit tests or other XSL functions contained this attribute, they cannot be parsed by Saxon. To correct this problem, use an expression such as `select=` instead of `value=`.

Migrating from Process Developer Versions Prior to 8.0.x

There are a few changes to be aware of for new versions of Process Developer.

- Deployments from prior versions are shown as legacy contributions on the server. Although there is no change in how your processes run on the server, we highly recommend that you delete old BPRs and redeploy your processes and resources to take advantage of contribution features. See *Converting Legacy Deployments to Contributions*.

- In the PDD, there are wording changes for Process Version details to match the new contribution states. *Effective* date is now *Online* date and *Expiration* date is now *Offline* date.

- There is a new Swimlane annotation in the Process Editor's tool palette (BPMN style only). To view this new Palette item, be sure to create a new workspace or delete your current Palette customization file.

- You can no longer import a WSDL declaration into your BPEL from the file system. You must import the WSDL into your project or set project references to include another project. Existing file system declarations are converted to URLs.

- The `.avcconfig` file for Process Central has been updated, but older files are compatible with Process Central. The new file allows for custom columns in Process Central.

Opening and Using BPEL4WS 1.1 Processes

If you have BPEL processes created in earlier versions of Process Developer, they conform to the BPEL4WS 1.1 specification. You can open and work with BPEL 1.1 processes. Process Developer recognizes the version of BPEL in a process and opens in the appropriate mode: BPEL4WS 1.1 or WS-BPEL 2.0.

In BPEL4WS 1.1 mode, Process Developer displays the palette and menu options that support the BPEL4WS 1.1 specification.

By default, Process Developer is set to launch in WS-BPEL 2.0 mode. You can set a preference to open Process Developer in BPEL4WS 1.1 mode, if desired. For details, see *Process Developer Preferences*.

Migrating Processes from BPEL4WS 1.1 to WS-BPEL 2.0

You can migrate any BPEL process you created in earlier versions of Process Developer, or any BPEL4WS 1.1 process, to a WS-BPEL 2.0 process. Doing so gives you access to all the new and updated activities, data manipulation, and handlers supported.

You can migrate your BPEL 1.1 processes as follows:

1. From the Project Explorer, double-click a BPEL file to open it in the Process Editor.

2. Select **File > Save As**.

3. In the **Save As** dialog, select another file location, if desired, and rename your BPEL 2.0 file if you do not want to overwrite the BPEL 1.1 file.

4. Select the checkbox next to Convert to WS-BPEL 2.0 format, as shown in the example, and select **OK**.
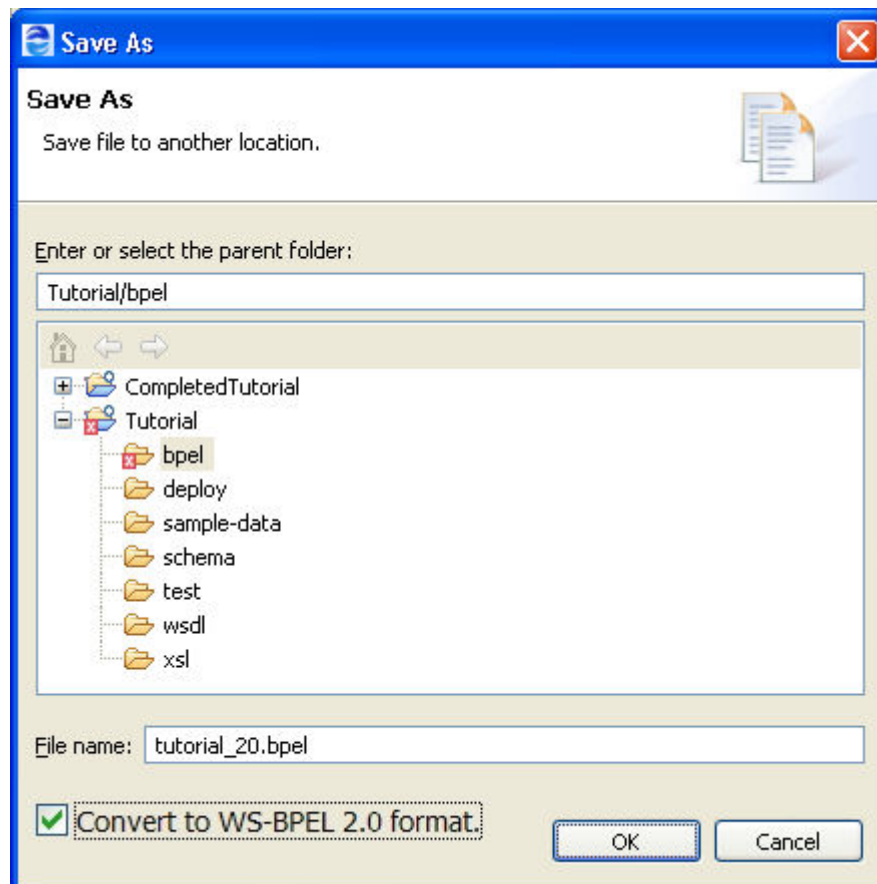
Your BPEL 2.0 process opens in the Process Editor, and Process Developer changes to WS-BPEL 2.0 mode.

All of your preference settings are preserved.

Conversion issues:

- Fault variable. The variable used in a catch activity is accessible only in the catch activity. In BPEL 1.1 the variable was declared in the enclosing scope. It is no longer available anywhere except within the catch. If a 1.1 process was using the fault `var` outside of the catch, the static analysis will report an error.

- OnEvent variable. Same issue as above. The variable is now declared within the `OnEvent` activity, not in the enclosing scope. The variable is accessible only to the `OnEvent` and not to the scope.

- Queries in copy operations. The absolute path used in 1.1 processes is now a relative path. However, a query using an XPath expression beyond a simple path to data may not get converted. A query that cannot correctly convert is left in `get Variable Data()` syntax. Process Developer warns you that the expression is not WS-BPEL 2.0 compliant.

Custom Function Pick Lists

The XML format for custom function pick lists has changed for Process Developer, for all process versions. You must update expression builder pick list files as follows.

The old syntax is:

```
<pickList xmlns="http://active-endpoints.com/FunctionBuilderSchema">
 <pickHeader ...>
  <pickItem pickDisplay="ncname"
      syntax="namespace:ncname"
      caretOffset="colnum-expr"
      hoverHelp="prefix|namespace:ncname Description:ncname"/>*
 </pickHeader>
</pickList>
```

The new syntax is:

```
<pickList xmlns="http://schemas.active-endpoints.com/picklist/
        2006/08/ExpressionBuilder_FunctionsPickList_Schema.xsd">
 <pickHeader ...>
  <pickItem pickDisplay="NCName"
      syntax="${prefix}:NCName(${caret})"
      hoverHelp="prefix|namespace:NCName Description: NCName"/>*
  </pickHeader>
</pickList>
```

If your Process Developer installation uses an existing workspace, your already-loaded custom function picklist will continue to work. However, you can not reload it without receiving a validation error.

Taking Advantage of SOAP 1.2 Port Binding Version 7.x

Process Developer Version 7 and later supports endpoint reference port bindings for SOAP 1.1 and SOAP 1.2. All processes prior to Version 7 supported only SOAP 1.1. If you want to take advantage of the SOAP specification upgrade, SOAP 1.2 clients should use the new SOAP 1.2 URL when addressing My Role endpoints, namely:

```
http://host:port/active-bpel/services/soap12/MyRoleService
```

Partner role services can be invoked with SOAP 1.2 bindings too.

Saving From One Editing Style to Another (BPMN and Classic)

Process Developer Version 7.0 introduces the BPMN editing style. This style is based on common icons and symbols from the Business Process Modeling Notation specification. You and your business partners may be familiar with the workflow-style icons and symbols widely used by business analysts and developers.

You can convert a process from one editing style to another if desired. A process that is created in Process Developer Classic editing style can be saved as a BPMN style and vice versa.

To convert from one editing style to another, open a process in the Process Editor and select **File > Save As**. In the **Save As** dialog, select the Editing Style you want: BPMN or Classic. For details, see *BPMN-Centric and BPEL-Centric Edit Styles*.

Converting Legacy Deployments to Contributions

Process Developer Version 8.0 introduces contributions. A *contribution* is a deployed business process archive (BPR), managed as a unit of files. Rather than deploy and replace individual BPEL processes and resources, you deploy both current and updated as a unit that is easily managed on the server. For details, see *Managing Deployment Contributions*.

If you are upgrading to a new version of Process Developer, note that all BPR deployments prior to Version 8 are preserved as legacy contributions. On the Contributions page of the Process Console, you will see one or more entries named `legacy:/bpr.deployment` The newest deployment has the status of *online*. Previous deployments have the status of *offline pending*.

We highly recommend that you delete old BPRs and redeploy your processes and resources to take advantage of contribution features.

Here are some recommendations for converting your old deployments:

- Delete old BPRs and BPRDs from a project. You will create just one new BPR/BPRD per project. The BPR treats all deployable project files as a unit that is easily managed on the server.
- (Recommended). In your project, create a folder to hold all non-deployment files, such documentation files and unused resources. This is recommended, but is not required. You can add this folder to a list of excluded folders for deployment. See *Contribution Preference* for details.
- (Recommended). If your project is using WSDLs from another workspace project, you can specify that project as a project reference to manage those WSDLs in a separate contribution. For advantages to this approach, see *Using Project References*.
- Create and export a new project-based business process archive. Notice that the new Export Wizard automatically includes all deployable resources in your project and excludes test and sample data files.
- Be sure to deploy all deployable files each time you redeploy, even if you have only updated one or two files. Deploying an updated individual resource creates an undesirable version of a contribution.

Note that as you redeploy your BPEL processes and associated resources, they are removed from the `legacy:bpr.deployment` and added as contributions.

C H A P T E R   2

Using Guide Developer for the First Time

This chapter includes the following topics:

Launching Process Developer

To launch Process Developer, use the shortcut button added during installation. Alternately, open your Process Developer installation folder, and in the Process Developer folder, locate `developer.exe` (Windows) or `designer` (Linux).

If you installed Process Developer as Eclipse plug-ins, start Eclipse and from the Window menu, select **Open Perspective > Process Developer**.

When you launch Process Developer for the first time, your workspace looks similar to the following:
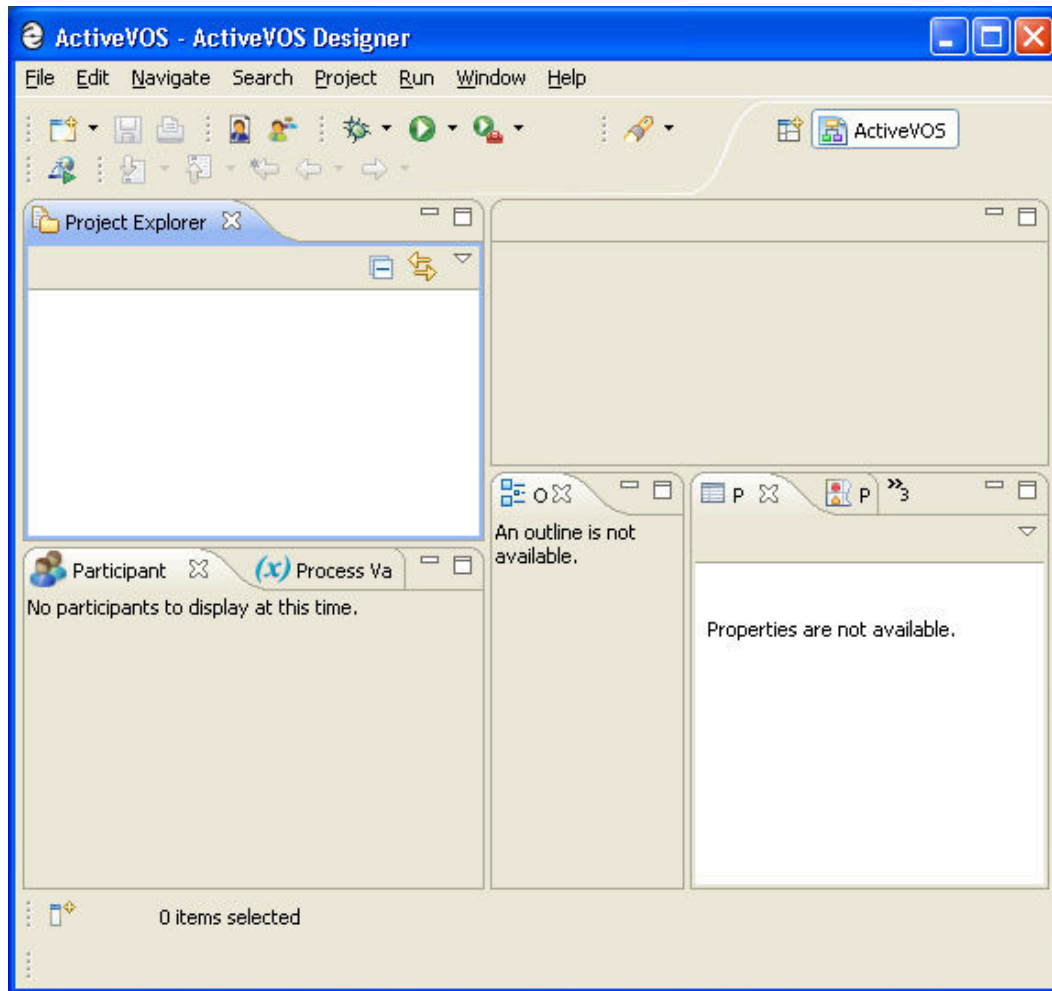
To get started, you can create the Tutorial Orchestration project or create a new orchestration project or import an existing project. It is recommended that you create a new project in the workspace folder that is the default location.

For more information, see *Orchestration Project Templates* and *Using the Workspace to Store Projects*.

Using the Workspace to Store Projects

The *workspace* is the folder on disk that contains all of your projects files, as well as meta-data such as preferences you may have customized. During installation you are asked where you want your workspace created.

All of the projects and files you create are added to the workspace folder and are displayed in the Process Developer Project Explorer view. If you make a change to a file outside of Process Developer, the change is reflected in Project Explorer.

For more information, see:

- *Creating an Orchestration Project*
- *Importing an Existing BPEL Process*

Creating an Orchestration Project

You must create a project folder to store BPEL process and related files. A project maps to a folder in the file system. A default orchestration project includes the following in support of BPEL orchestrations:

- **Service References**. For details, see *About Interfaces, Service References, and Local WSDL*.
- **Folders**. Several folders are added for an organized and convenient registry of the resources that support orchestrations.
- **Builders**. For details, see *About Project Orchestration and Validation Builders*.

In addition, there are several templates available to help you get started on your projects. See *Orchestration Project Templates*.

To create a new orchestration project:

1. Select **File > New > Orchestration Project**, and click **Next**.
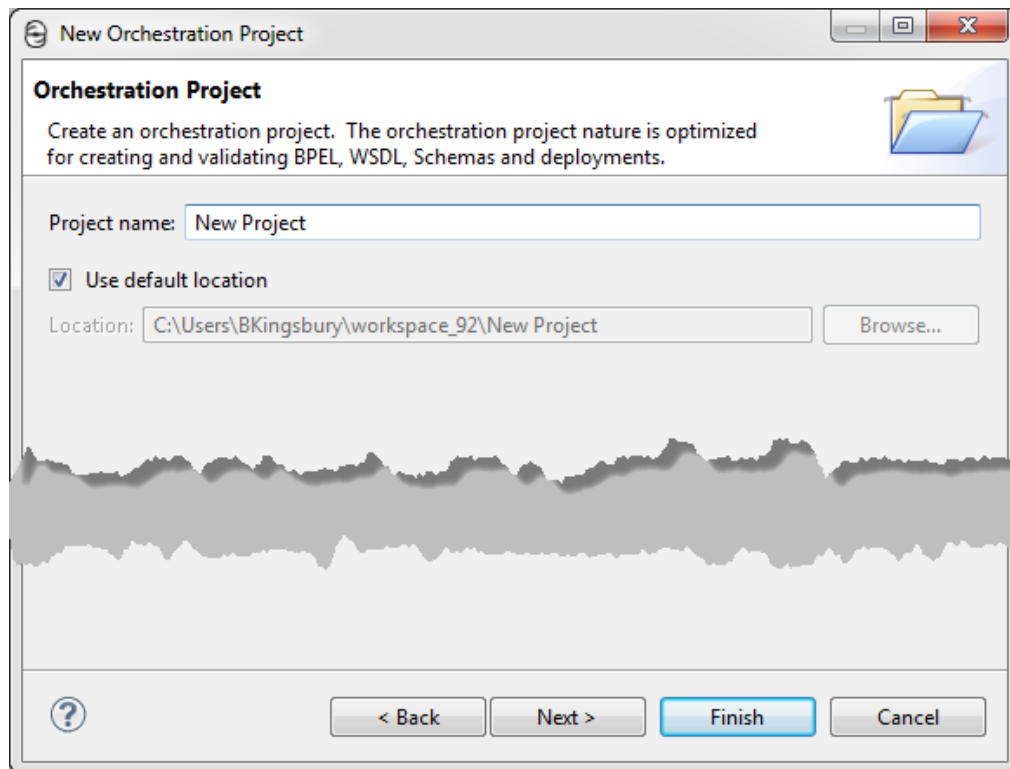
2. Type in a project name.
3. Do one of the following:
   - Click **Finish** to create the new project.
   - If desired, deselect the **Use default location** check box. For details, see *Changing the Default Location of a Workspace Project*.
   - Click **Next** to select a template for a special type of orchestration project. For details, see *Orchestration Project Templates*.
4. Notice that by default, several folders can be created to store process files and related resources. These folders are provided by the orchestration project's nature.
5. Your new project appears in the Project Explorer, and the folder in the file system contains a *.project* file.

You can now create new BPEL process files, as described in *Creating a New Process*.

See also, *Adding or Removing a Project Orchestration Nature*.

Changing the Default Location of a Workspace Project

When you are creating a new orchestration project, you can deselect the check box next to Use default location and browse to a file system location to create a project.

When selecting a non-default location:

- If the file system folder you select contains subfolders, all subfolders become part of the project. Be sure to select **Make New Folder** and type in a folder name if you want an empty project, as the illustration shows.

- You cannot create nested projects. Be sure that your projects do not overlap.

Orchestration Project Templates

The Orchestration Project templates provide built-in resources and natures to choose a starting point for your project.

When you select **File > New > Orchestration Project**, and name a new project, you can click **Next** to select a specialized orchestration project template.
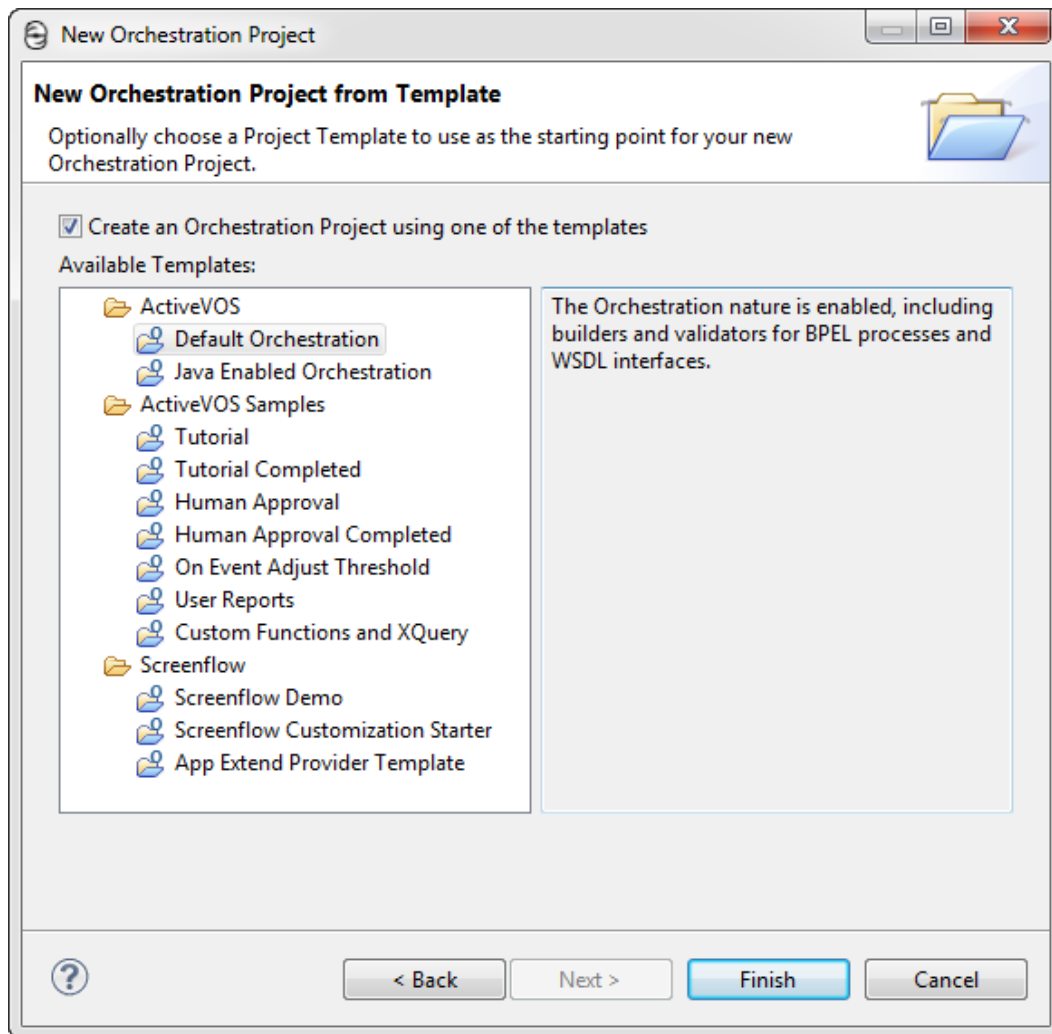
The templates contain files, resources, and orchestration natures (see *Adding or Removing a Project Orchestration Nature* for information) to help you get started on special projects. For example:

- **Java Enabled Orchestration**. For details, see *Creating a Java Interface*.
- **Tutorial**. The tutorial contains WSDL and samples. It is the starting point for creating your first BPEL process.
- **Tutorial Completed**. A complete BPEL process plus all related resources for simulation and deployment

Other project templates help you discover ways to use Process Developer.

Adding or Removing a Project Orchestration Nature

If you import or create a project in the Project Explorer, you can add an *orchestration nature* to the project. An orchestration nature provides enhanced creation and validation of orchestrations.

With an orchestration nature, you can:

- Import service references and view WSDL port types, partner link types, messages, and sample data. For details, see *Importing a Service Reference*.
- Use automatically created project folders for all of your required resources.
- Validate your files, as described in *About Project Orchestration and Validation Builders*.

To add an orchestration nature, right-mouse click on a project and select **Add Orchestration Nature**.

If the orchestration nature already exists, you can select **Remove Orchestration Nature**. The project is converted to an Eclipse project.

About Project Orchestration and Validation Builders

As you add and modify BPEL processes, a background validation builder works to ensure locate errors and problems in your process, imported WSDLs, and other resources. Errors, warnings, and information are reported in the *Problems* tab.

Each orchestration project has two builders:

- The **orchestration builder** performs validation on BPEL, B-unit, PDD and PDEF files. This builder also tracks dependencies between files. For example, if a WSDL file changes, the BPEL file importing it is automatically validated.
- The **validation builder** is part of the Eclipse Web Tools Project that ships with Process Developer. This builder validates WSDL and schema files and checks for WS-I conformance.

To disable or enable builders for an orchestration project:

1. In the Project Explorer, right-mouse click on an orchestration project, and select **Properties**.
2. In the Properties dialog, select Builders.
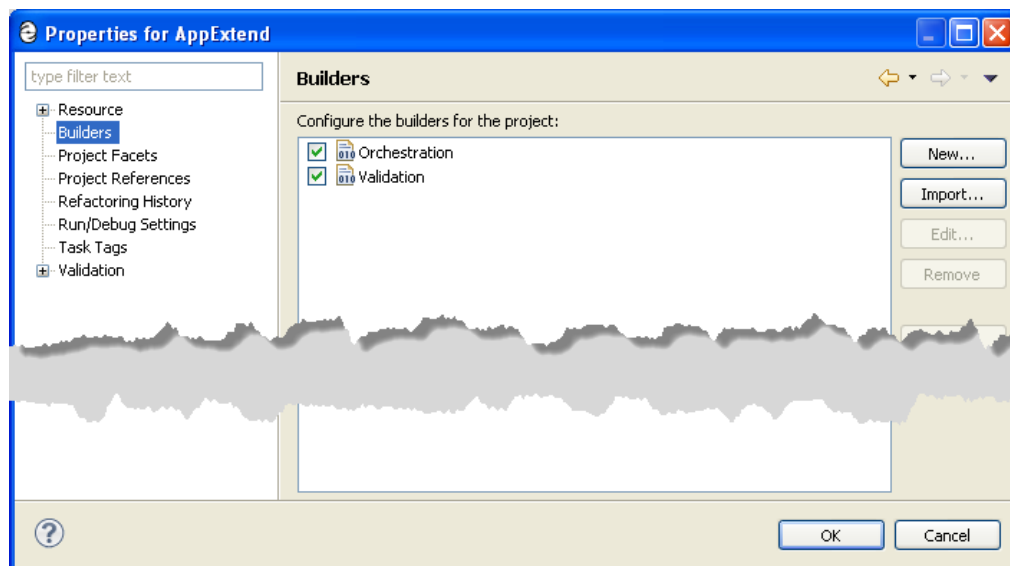
Notice that the Orchestration and Validation builders are enabled, and you can disable them if desired.

You can set a preference for WSDL cache and time out values to make building faster or slower, depending on the speed of your computer or network. See *Cache and Timeout Preference*.

Using Project References

As you create a project, you can keep common WSDL and schema files in another workspace project. Using this technique allows you to maintain and deploy commonly used WSDLs separately. It allows for the best reuse of common resources.

Many resources can be shared between multiple processes. The most efficient practice is to separate them out and place them in one or more resource-only projects. These can then be deployed independently as their own contributions. The process projects can declare dependencies on these resource projects, and are deployed after the resource-only projects they depend on.

You can define another project as a dependency for your current project using Project References.

1. Before beginning a new BPEL process, create a new orchestration project to store common WSDL and schema.

2. In the Project Explorer, right-mouse click on the project name of a project that will contain BPEL files, and select **Properties**.

3. Select **Project References**.
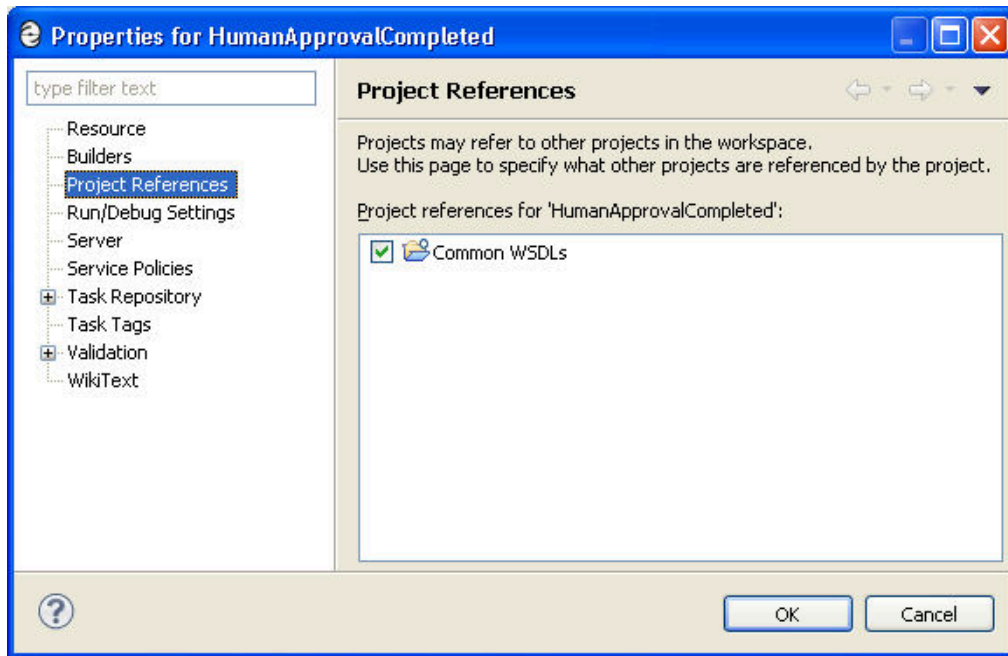
4. Select a project from the list that contains resources you want to reference. The example shows a project named "Common WSDLs":

5. When you create a new participant, the WSDLs are available, and are displayed in a Project References folder. For details, see *Using the Participants View*.

See also *Deploying Project Dependencies and Viewing Excluded Dependencies*.

Creating a New Process

When you create a new process, you add the BPEL file to a workspace project.

Ensure that at least one orchestration project is displayed in the Project Explorer before creating a new BPEL process. For details, see *Creating an Orchestration Project*.

Beginning a New Process

The following procedure shows how to begin a a new process:

1. Select **File > New > BPEL Process**.

2. For organization and convenience, select the `bpel` folder in your orchestration project folder, and in the *File name* field, type in a name for your BPEL file. The `.bpel` extension is automatically added.
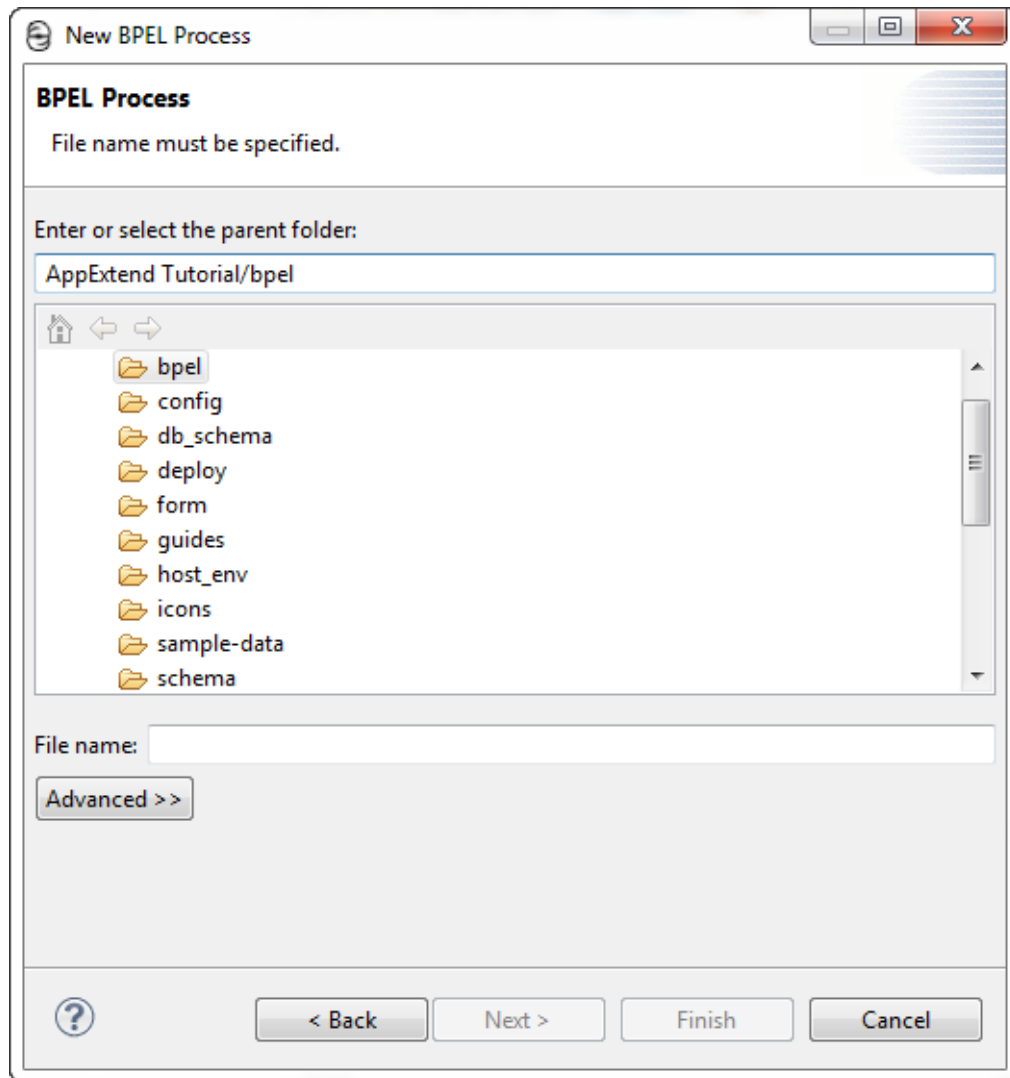
3. Select **Advanced** to view properties that you can set for this process. These properties display below this **Advanced** button.
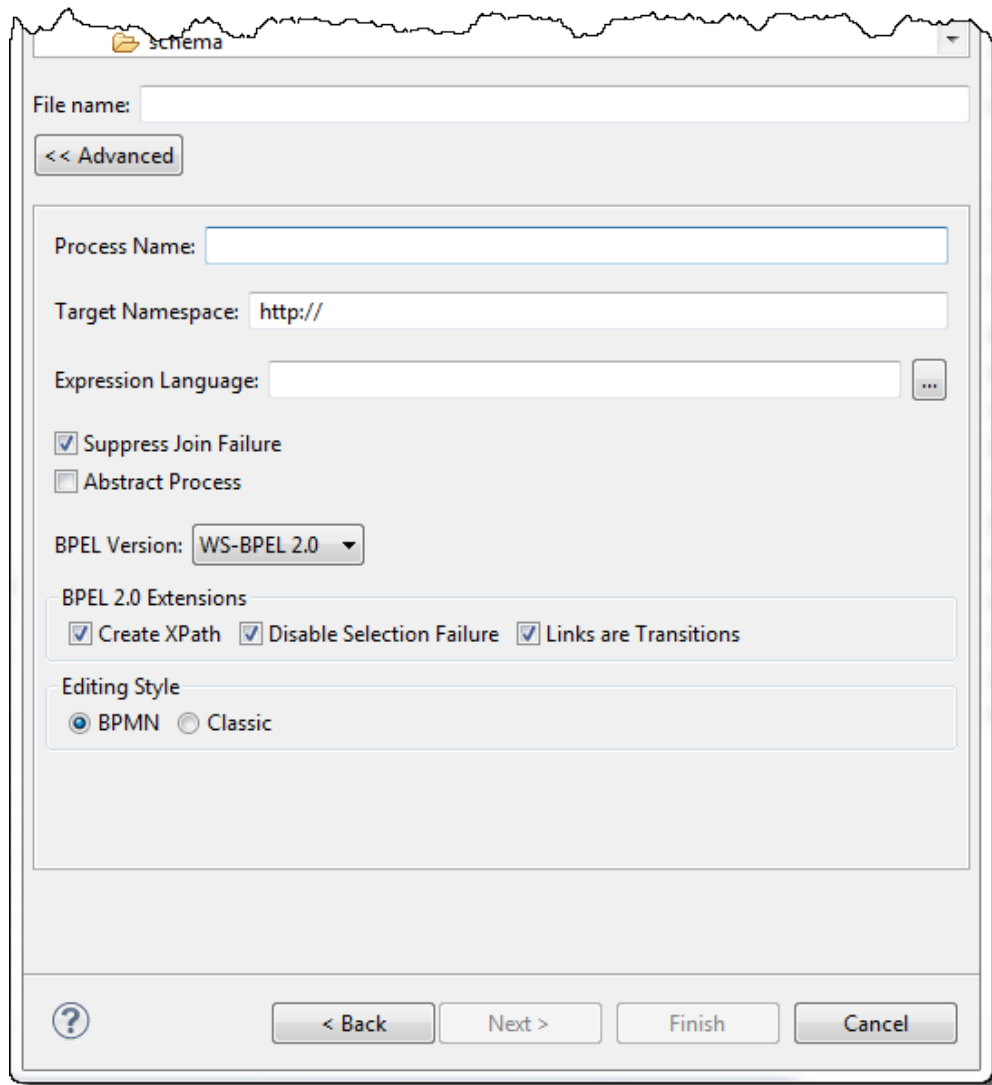
The default values (for the check box properties) are shown and can be set for all processes within the *Process Developer Preferences*. In addition, you can set the process name, target namespace and other properties for this process later. See *Process Element and Properties* for more information.

Click **Next** after filling in this dialog.

4. Select the BPEL Templates page to create a special purpose process. For details, see *Using a BPEL Template for a System Service-Based Process*.
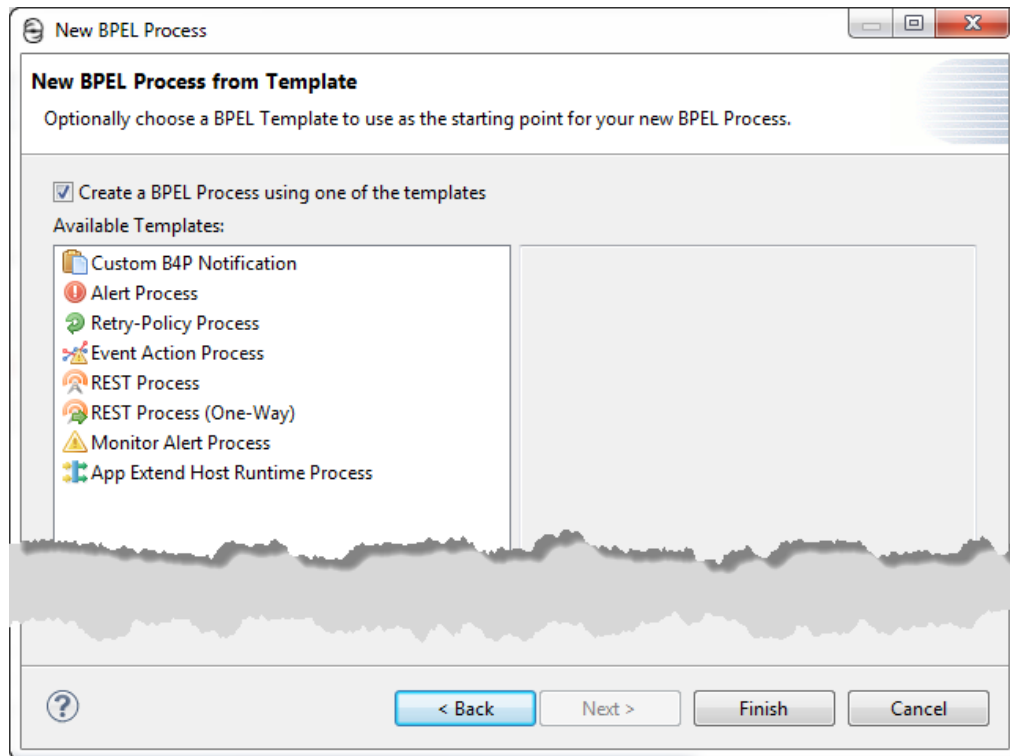
5. Click **Finish**.

Your BPEL file opens on the Process Activities tab of the Process Editor.

Notice that an error icon appears next to your folder. This icon indicates validation of your BPEL file has occurred, and that your process is not currently valid. This icon disappears as you add the necessary activities to the process.

For details on validation, see *About Project Orchestration and Validation Builders*.

Selecting the BPMN-Centric or BPEL-Centric Palette

A BPMN palette contains standard icons familiar to Business Process Modeling Notation users. A BPEL palette uses many BPMN icons within a BPEL terminology context. The same BPEL source code is generated no matter which palette you use.

A BPMN palette contains standard icons familiar to Business Process Modeling Notation users. A BPEL palette uses many BPMN icons within a BPEL terminology context. The same BPEL source code is generated no matter which palette you use.

You can select the set of icons to display on the editor palette:

- **BPMN-Centric**
  Uses standard Business Process Model and Notation elements
- **BPEL Centric**
  Uses BPMN notation within a BPEL naming context

Set the way Process Developer displays icons within the Editing Style area within the **New BPEL Process** dialog. You can also set a workspace-wide preference for a palette style in *Layout Preferences*.

For details, see *BPMN-Centric and BPEL-Centric Edit Styles* and *Which Edit Style to Choose: BPMN-Centric or BPEL-Centric?*.

Importing an Existing BPEL Process

If you have created a BPEL process outside of Process Developer, you can import the BPEL and related files into our orchestration project. If your BPEL process in already in an Eclipse project, see *Adding or Removing a Project Orchestration Nature*.

Here how you do this:

1. Select **File > New > Orchestration Project**, and click **Next**.
2. Type in a project name, and click **Finish**.
3. If you have XSD files, select the schema folder, and right-mouse click to select *Import*.
4. Select **General > File System**.
5. Browse to the directory where your XSD files reside. Import your XSD files into the schema folder.
6. Import your WSDL into the `wsdl` folder, or alternately use Service References, as described in *Importing a Service Reference*.
7. Import your BPEL files into the `bpel` folder.
8. Double-click the BPEL file to open it in the Process Editor.
   The file opens in the mode associated with the BPEL version: WS-BPEL 2.0 or BPEL4WS 1.1. It is opened in the modeling style, either BPMN, BPEL, or Process Developer classic, associated with the style sheet it was created with.

   For details on saving your file with a different style sheet, see *BPMN-Centric and BPEL-Centric Edit Styles*.
9. Use the Problems view to correct any validation errors that occurred.

If you import your BPEL process before you add the referenced WSDL and schemas, Process Developer reports validation errors until it locates a namespace for your process.

If your WS-BPEL 2.0 process contains elements and attributes that are extensions to the WS-BPEL 2.0 specification, they are automatically declared. See *Declaring Extension Elements and Attributes* for details.

Importing a Visio XML Drawing File

The Visio drawing used in your workspace can be based on BPMN shapes. You import it into a workspace project.

You can import a Visio XML drawing from Visio 2010 Premium (or later) that is created with the BPMN template or flowchart using BPMN shapes. Other types of Visio drawings cannot be converted to BPEL.

To import the file:

1. Save it as an XML drawing with the .vdx extension.
2. Select **File > Import > Orchestration > Visio XML Drawing**
3. Select a Visio file from your workspace or the file system. The file must have the .vdx extension.
4. Select a project folder location for the file. For example, select the "bpel" folder of an orchestration project.
5. If needed, type in a name (or keep the default) for the files being converted to BPEL. You may need to supply a name if the file includes multiple pools. If a Visio file contains multiple pools, each pool is converted to one BPEL file. Multiple pools are not supported in one BPEL file. Process Developer automatically adds the .bpel extension.
6. If needed, you can overwrite an existing BPEL file.

Tips on converting Visio files:

- The Visio file is not imported into your project. The import wizard converts the Visio file to BPEL.

- The Scale factor creates the BPEL process with a default amount of space between activities. The scale factor is a Visio measure unit (by default in inches) to convert to pixels in Process Developer. If the default of 150 creates a BPEL process that displays too much space between activities, try setting the value to 100. Alternately, in Process Developer, select **Auto Layout** to improve the placement of activities on the canvas.

- Messaging data that exists between pools is not supported. In Process Developer, you must create a process-to-process or process-to-subprocess invoke handler in the Process Deployment Descriptor.

- Be sure to view the Errors and Problems views to see messages generated from the conversion

Begin Your First Project with Process Developer Assistance

Process Developer offers many entry points into building your first process. Here are some pointers to help you decide how to begin.

My Knowledge, My Resources	Start Here
I'm a newbie	Use the *Hello World* examples in the Process Developer Developer's Education Center on `http://www.activevos.com`
I want to learn how to use Process Developer end-to-end	Walk through the Process Developer Tutorial.
I don't have any WSDL but I know what I want the process to do	See *Using the Participants View* to generate the process and service roles and participants
I have sample data files (XML files) that represent the input and output of the process I need to build	See *Using the Participants View* to generate a service and service participant
I have WSDL	Import your WSDL and use the Participants view to build the service participants and activities

Setting Up the Embedded Process Server

The Process Server consists of the server engine running under Apache Tomcat. Tomcat is the servlet container that is used in the official reference implementation for the Java Servlet and JavaServer Pages technologies.

When you create a BPEL process and its deployment descriptor, you can deploy process resources directly to a running server. This allows you to execute and remotely debug your processes.

To set up the server:

1. Select the Servers view in the lower right of the workspace.

2. Right-mouse click within the *Servers* view and select **New > Server**.

3. In the Server type list, select *Process Server*, and select **Next**.

4. If the default HTTP and connector ports are already in use on your computer, you can change them on the Ports page and then click **Finish**.

The Console shows the Tomcat start-up activity. At the end of start-up, you'll see a "server started" message for Tomcat.

To start the server:

After Tomcat is started, select the **Start the Server** icon in the *Servers* view for the Process Server, as shown.
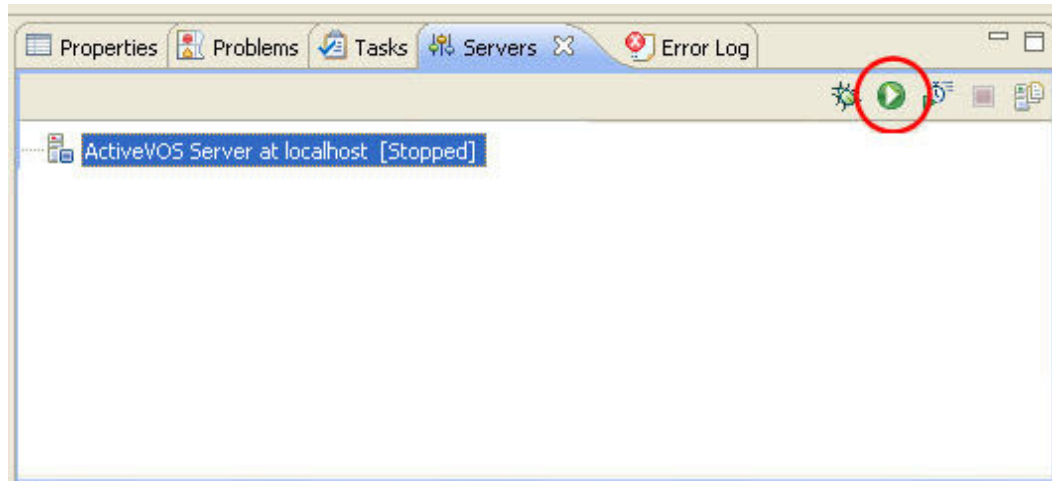
To view the Process Console, select the **Process Console** icon in the Process Developer toolbar.

You can also view the console in a browser. Do this be going to the Console's URL. The default URL is `http://localhost:8080/activevos`.

To configure https for the Embedded Process Server, follow the steps below:

- Manually add the connector for https port in the server.xml of the embedded tomcat in the following location: `org.activebpel.enginep_9.2.5 under the installation_dir/designer/plugins`

- Follow the steps below to add the port details in the Tomcat launch configuration:

  1. Create Tomcat Embedded server.

  2. Open configuration page and doubleclick ActiveVOS Server at localhost.

3. Click **Open launch configuration** from the Overview page as shown below:
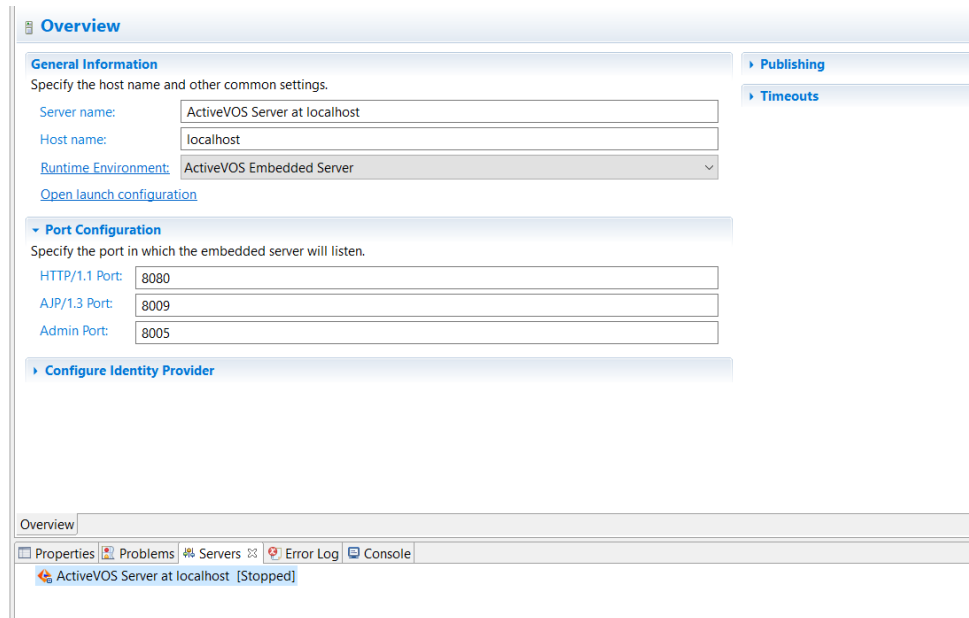
4. Add the following environment variable in the Arguments tab: Dae.web.filter.https.port=8443, as shown below:
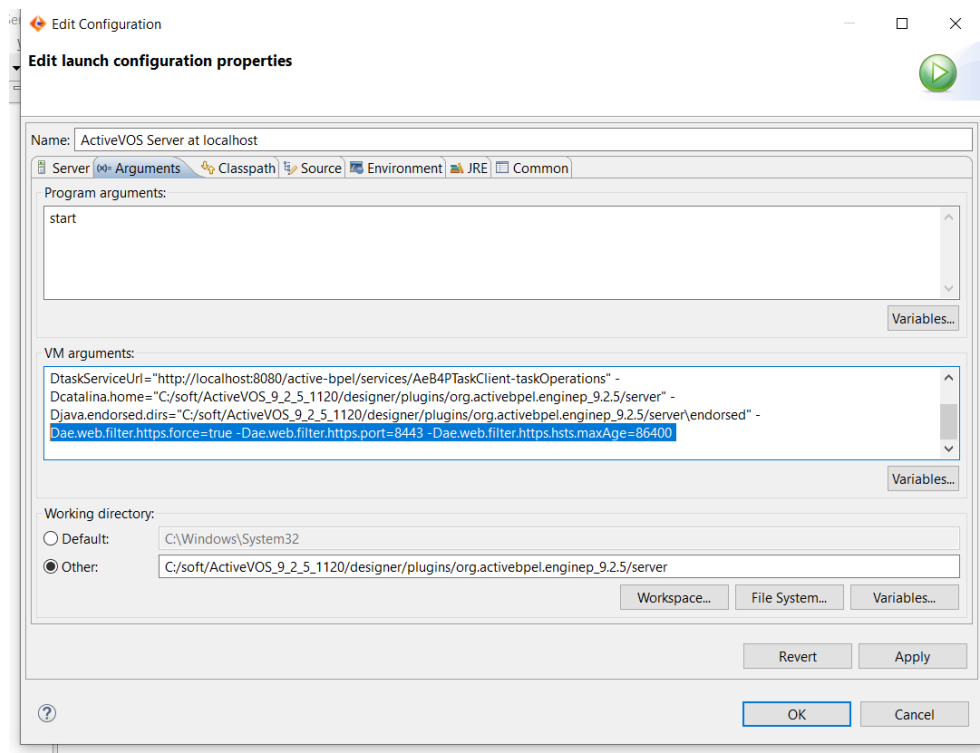
Process Developer Orchestration File Resources

As you develop orchestration projects, you accumulate many types of files, including BPEL, WSDL, XSD, HTML, XML, and others. In addition to these, Process Developer generates some project files that are hidden by default. The following is a list of resources that Process Developer can add to your project. Based on the descriptions of these files, you can decide if you want to add them to your source control repository.

Process Developer Resource	Description
`.image`	When you save a process, a screenshot of your process is automatically saved as multiple images in a folder named `.image`. When you deploy, Process Console uses these images to display the Detail Graph View of the process. Without this information, a default process graph is displayed that may not match the look of your process. We recommend that you preserve the `.image` folder and all files in it.
`.archive_settings.properties`	In the BPR Export wizard, you can make selections for the Web Service URL and other settings. These settings are stored in `.archive_settings.properties`. During subsequent uses of the BPR Export wizard in a project, the Process Developer uses information in this file.
`merge_mappings.xml`	(Human Tasks) When you create a task form for a People activity, you can map task input data to output data fields so that a consolidated list of fields are displayed in the form for the input and output fields that could be merged. The input-output mapping is stored in this XML file. Typically a mapping is created only through a one-time use; however, we recommend you preserve this file with your project.
`.BPR`, `.BPRD`, `.project`, `.classpath`	Business process archive files (BPRs) can be regenerated by using the Export wizard. However, if preserved, they are easily shared among team members because they are self-contained. An Ant file that you generate (`.BPRD` file), on the other hand, can contain a Web URL that points to a local host and port that other team members may need to modify. The Eclipse files created, including `.project` and `.classpath` (if used), should be preserved with your project.

CHAPTER 3

Getting Started with Informatica Process Developer

This chapter includes the following topics:

Using Source Control Systems

Process Developer can be used with source control management systems. CVS is built into the Process Developer installation package. If you would like to use the Subversion source code management, here's how to install it:

1. Click **Help > Install New Software**.
2. Click **Add**.
   Enter a Name, which is *Subclipse Update Site*.

   Enter the Location, which is http://subclipse.tigris.org/update_1.6.x
3. Click **OK**.
4. Select all items in the list.
5. Click **Next**.
6. Click **Next** again.
7. Select the *I accept* radio button.
8. Click **Finish**.

Eclipse will now install SVN.

If warnings appear, just accept them. You should also restart Process Developer if your are prompted to.

Navigating Through Process Developer

The Process Developer user interface consists of various perspectives, with associated views, editors, and menus.

Windows Perspectives Views and Editors

The Process Developer uses the basic Eclipse Workbench user interface components: windows, perspectives, views, and editors.

Each time you launch Process Developer, the Workbench window opens and displays a default perspective. A *perspective* consists of views, editors, menus and toolbars that support a set of tasks.

Each perspective has several views. A *view* is a unique window, designed for one task, such as displaying an XML tree, listing tasks, or showing object properties. Each view has its own toolbar.

Each perspective also has an editing area where files are opened. An *editor* is associated with a file type and provides the appropriate file actions. The most common file type is a document that can be edited, but some file types require special actions, such as automatically generating BPEL XML code. Editors share the same real estate, and are stacked on top of each other when several files are open. Many editors have a marker bar where you can add bookmarks, tasks, and breakpoints, as needed.

Here are some tips for customizing your work environment:

Editor Tips	View Tips	Perspective Tips
Double-click the titlebar to make the editor full screen	Hide a view by right-mouse clicking on the view titlebar and selecting *Fast View*. The view is moved to the shortcut bar as an icon. Click the icon to open the view.	Select **Window > Customize Perspective** to show and hide views.
Tile two or more editors by dragging one titlebar to the left edge of the editing area	Rearrange views by moving and docking them and by rearranging their tabs.	Open the same perspective in two windows by selecting **Window > New Window**.

Process Developer Perspective

The Process Developer perspective is the default product perspective and contains the views, editor, menus, and toolbars that support the tasks for designing and generating a BPEL process definition.

When you first launch Process Developer, you see its default window layout, called a *perspective*. A perspective consists of views, editors, menus and toolbars that support a set of tasks.

A *view* is a unique window within a perspective, designed for one task, such as displaying an XML tree view of a file, listing errors or tasks, or showing object properties.

Process Developer contains two default perspectives:

- Process Developer Perspective
- Process Developer Debug Perspective

Perspectives are versatile layouts that you can customize. See *Customizing the Process Developer Perspective* for details.

You can open and close perspectives using their icons on the Perspective Fast /View bar. The default position for the perspective bar is top right. As the following illustration shows, you can dock the bar in another location, shown here on the top left.
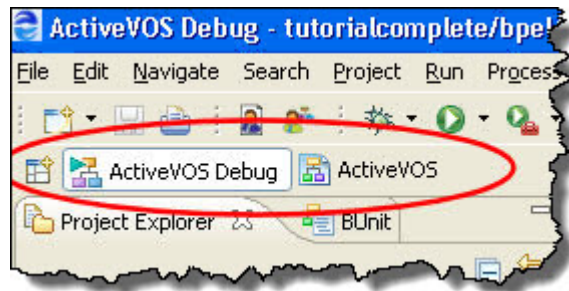
Project Explorer

The Project Explorer displays projects, folders, and files located in the Process Developer workspace folder. You must create a project before creating a BPEL process. An enhanced project type for BPEL is the orchestration project. For details, see *Creating an Orchestration Project*.

The Project Explorer's orchestration projects contain rich details and builders for working on BPEL processes and related WSDL, schema, sample data, test, and deployment files.

The Project Explorer is a mirror of the file system's Workspace folder.
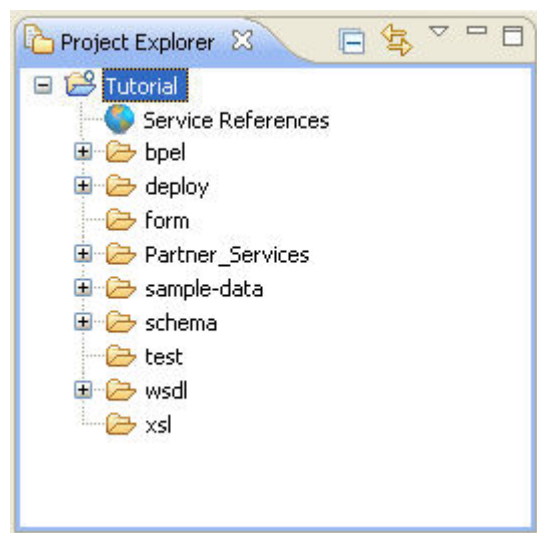
We recommend you store all files within orchestration projects.

Tips for using the Project Explorer:

- Add projects and files to the Project Explorer by dragging them in from the file system and adding an orchestration nature, which is described in *Adding or Removing a Project Orchestration Nature*.

- Import projects and files. For more information, see *Importing existing projects* and *Importing resources from the file system* in the Workbench User Guide Help.

- Copy and paste files from the file system.

- Add files to the project workspace in the file system and then use the **Refresh** command to update the Project Explorer. For more information, see *Workbench* and *Editing files outside the Workbench* in the Workbench User Guide Help.

- Use the many tools in the Project Explorer to filter, show, hide, and navigate through files. For more information, see *Project Explorer view* in the Workbench User Guide Help.

Participants

For details, see *What are Participants?*

Interfaces

By default, this view is not displayed. Instead, Participants view is displayed. You can display Interfaces by selecting **Window > Show View > Interfaces**. For details, see *Interfaces, Service References, and Local WSDL*.

Outline View

The Outline view displays all major components of a BPEL process, as shown.
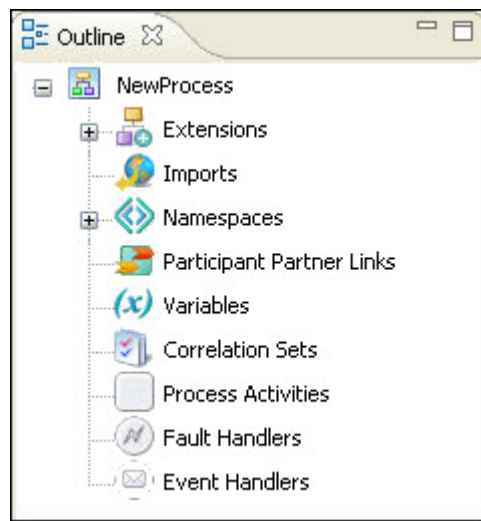
The nodes in the outline synchronize with the Process Editor and the Properties view. For example, select an activity in the Process Activities node to display the highlighted activity on the Process Editor canvas.

Tips for using the Outline view:

- Right-mouse click on an item to add new items of the same type.

- Reorder items by moving them up and down.

- Reorder variables to reorder them in the *Process Variables* view.

- Delete items you are not using.

- Double-click variables and other components to open the main property, such as the **Definition** dialog.

Activities are added to the Outline in the order they are created, and the BPEL XML code reflects this order. You may find it helpful to reorder the Process Activities node into activity execution order.

Problems View

The Problems view displays errors, warnings, and information related to the validation of your BPEL process. When you save your file, Process Developer performs static analysis on the BPEL file. This analysis validates the BPEL definition, expression language syntax, and WSDL references of the process, and adds items for invalid activities to the Problems view. Before testing your process, you can correct invalid activities.

As you correct errors and warnings, the related items in the Problems view disappear. You can also set a preference for levels of information you want to see in the Problems view.

The following illustration shows several errors and one warning for BPEL processes in Process Developer.
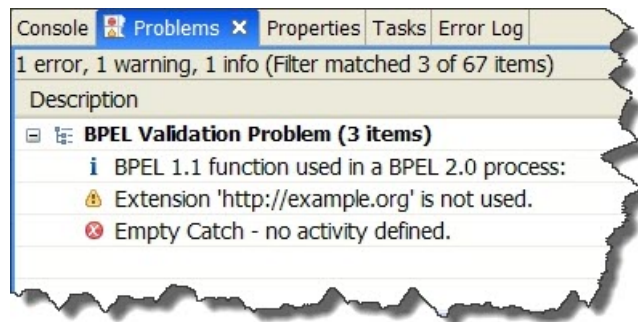
Tips for using the Problems view:

- Double-click an item to go to the associated resource.

- Right-mouse click an item to view property details or to select **Go To** resource.

- **Group** problems by type or severity.

- Use the extensive **Configure Contents** feature.

Tasks View

The Tasks view displays to do items you add. Add an item from this view, or right-mouse click on an activity on the Process Editor canvas.

The following illustration shows an example of the Tasks view.
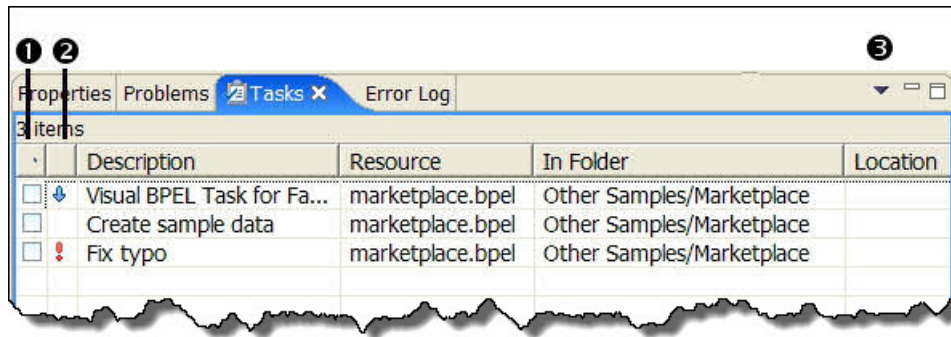
1	Check box for a to do item, indicating complete or incomplete. Check the box when the item is complete. Click the column header to sort by completion.
2	Priority for a to do task, including high, normal, and low. Click the column header to sort by priority.
3	Select display, sort, and filter criteria

Tips for using the Tasks view:

- Double-click a task to go to the associated resource.
- Right-mouse click a task to view property details or to select **Go To** resource.

Thumbnail View

If you design a complex process, it can outgrow the canvas display quickly. You can display the Thumbnail view to make navigation in a large process easier.

The Thumbnail view shows a miniature representation of your process. The section of the process currently showing in the Process Editor is shaded by a blue box. When a complex process cannot be fully displayed, move the blue box in Thumbnail view to display a selected area.

To display the Thumbnail view, select **Window > Show View > Thumbnail**.

To view another section of the process, click on the Thumbnail view shaded box, and move the box. The corresponding section of the process on the canvas moves into view. The following image shows this shaded box:
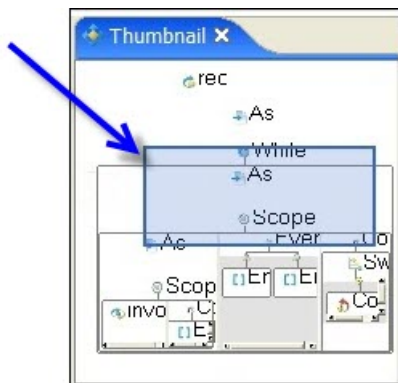
Bookmarks View

The Bookmarks view displays a list of bookmarks that you added to a BPEL process or a service file.

Add a bookmark to a BPEL activity or a service file to create an easy way to link back to the information. You can select a bookmark from the list and go to the place where you added the bookmark.

In the default perspective, *Bookmarks* view is not shown. To open this view, select **Window > Show View > Bookmarks**.

The following illustration shows an example of the Bookmarks view.
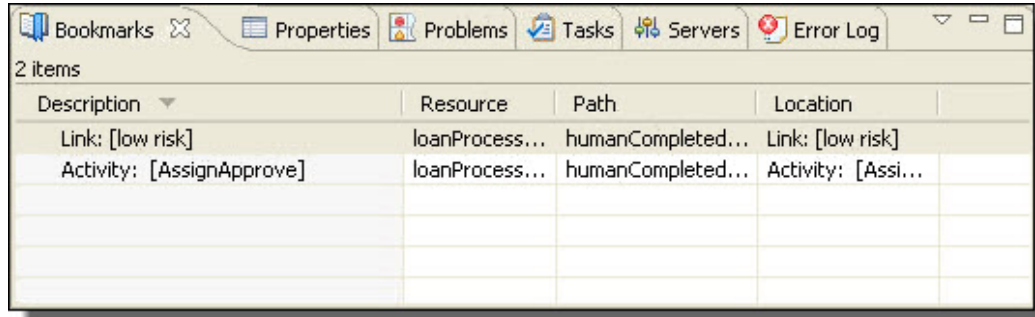
Double-click a bookmark to highlight its location on the Process Editorcanvas. For more details on bookmarks, see the *Workbench User Guide* help.

Relationships View

Open this view by selecting **Windows > Show Views > Relationship View**.

This view shows you a graphical representation of dependencies and references of BPEL, WSDL, XSD, and other artifacts in relation to the file open in the editor. Using this view, you can track the location of WSDL and XSD imports and other artifacts that are in the workspace.

For example, if a BPEL file is open, you'll see a relationship diagram of the files that the BPEL file depends on, such as WSDL, XSD, and HTML. You can select another file to see its dependencies and references.

You can toggle the view by using the toolbar buttons, **Show Dependencies** and **Show References**.

Servers and Console Views

The Servers view allows you to configure and start up the Process Server that is part of the Process Developer installation. You can deploy processes to the server and then execute them and remote debug them.

When you select **Start the Server** in this view, you can see the start up messages in the Console. The Console view shows messages from the Tomcat servlet under which Process Server runs.

For details of starting up and using the embedded engine, see *Setting Up the Embedded Process Server*.

Status Bar

The Status Bar shows the information about current actions and error messages. The following illustration shows an example of an error message in the Status Bar.

--- Variable Name must be unique.
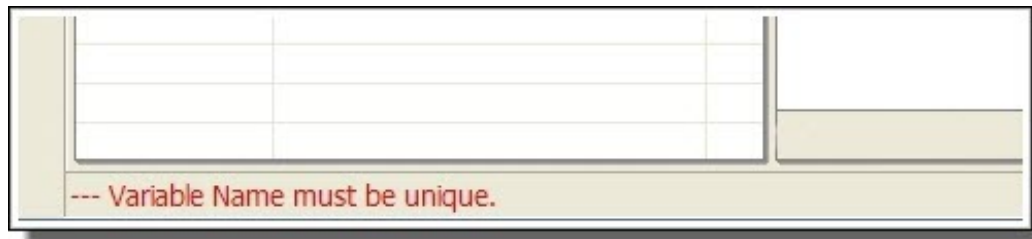
Guide Developer Debug Perspective

The Process Developer Debug Perspective contains the views, editors, menus, and toolbars that support simulated execution, B-unit testing, and remote debugging of BPEL processes.

The Process Developer Debug Perspective contains the views, editors, menus, and toolbars that support simulated execution.

For more information, see *Simulating and Debugging*.

BPMN-Centric and BPEL-Centric Edit Styles

Process Developer offers two edit styles for designing BPEL processes: Business Process Model and Notation (BPMN) and a BPEL-centric version of BPMN.

You can select the Process Editor Canvas edit style in *Layout Preferences*.

- **BPMN-Centric** edit style includes the graphical notation elements described in the Business Process Model and Notation V2.0 standard.
- **BPEL-Centric** edit style includes all BPEL constructs as well as some, but not all, graphic elements commonly used in BPMN modeling.

In both edit styles, every BPMN construct is serialized into a BPEL construct so that your process is 100% executable in BPEL.

New processes are associated with the edit style specified in Layout Preferences. Legacy processes are opened in the style they were created in.
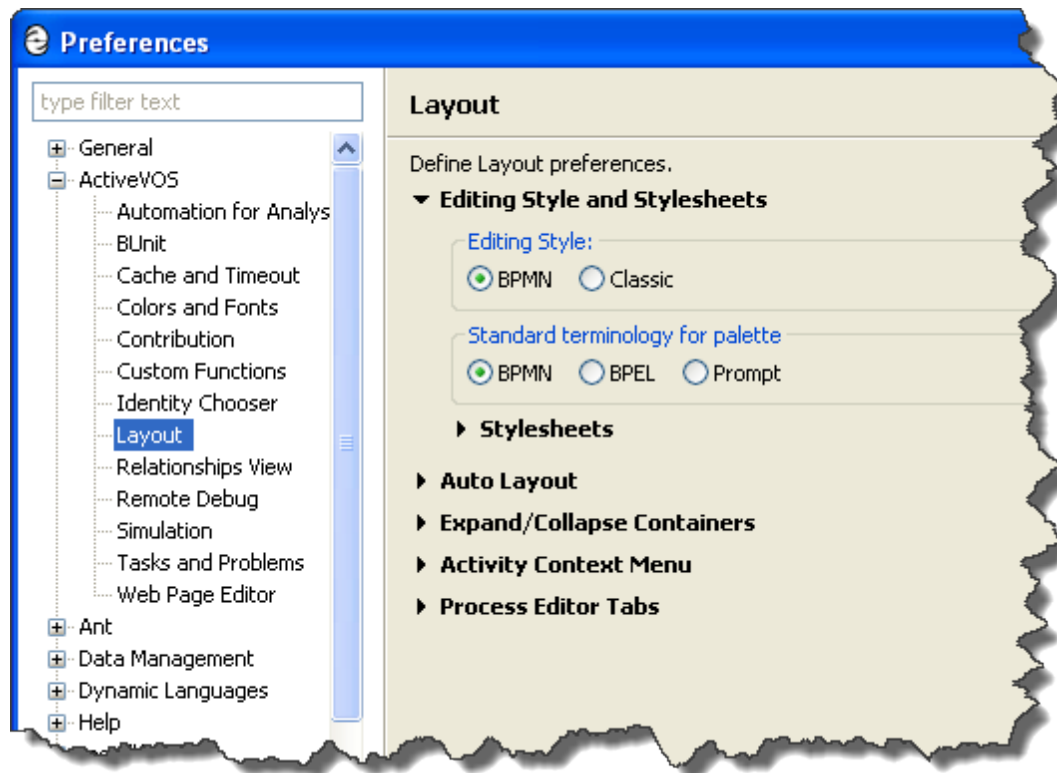
For legacy processes, the original **Process Developer Classic** edit style is supported. For details, see *What is Process Developer Classic Style?*

What is BPMN-Centric Style

Business Process Model and Notation (BPMN) is a graph-oriented notation standard favored by business analysts and designers to model the flow of activities for analysis, documentation, and execution. According to the standard, BPMN creates a standardized bridge for the gap between business process design and process implementation. Also, BPMN aims to ensure that BPEL can be visualized with a business-oriented notation.

The BPMN 2.0 standard is governed by the Object Management Group (OMG).

Usage Rules for Shapes and Symbols

BPMN uses shapes, symbols, and connectors to represent constructs. The shapes, symbols, and connectors have standard usage rules and standardized operational semantics. BPMN also includes extensive annotation and documentation capabilities.

The main shapes are:

- Rounded rectangle (activities)
- Diamond (gateways)
- Circle (events) with event-driven border styles (solid, thin, thick, dotted)

For a complete description of BPMN elements, see *BPMN-to-BPEL Implementation of Tasks and Events* and *BPMN-to-BPEL Implementation of Gateways and Control Flow*.

What is BPEL-Centric Style

BPEL-Centric style includes BPMN graphical notation to represent BPEL constructs. The BPEL palette contains the following notation:

- Each BPEL construct is in the palette, except flow and sequence. These constructs are represented implicitly on the canvas for efficient process design. For example, if you drag a receive to the canvas, it is automatically enclosed in a sequence. When you add the next activity, it is automatically linked.

- Some BPMN elements are included. To add visual clarity to a process, you can use the start, end, and none events and the fork-join and gateway control flows.

- Some BPEL activities, such as a receive, can be shown in more than one BPMN construct. For example, a receive can be shown as a message catch event or a receive task. In the receive's Properties view, from the Show As list, select the BPMN display style.

Special Usage of BPEL and BPMN Constructs

The following table shows the BPEL constructs hidden and BPMN constructs used in BPEL edit style.

BPEL Construct Hidden in BPMN	BPMN Construct (not part of BPEL terminology)
Flow (automatically embedded in a process design as applicable)	Fork Join
Sequence (automatically embedded to structure a group of activities)	Gateway
Opaque (not used)	Start/End/None

See also:

- *Comparing the BPMN-Centric and BPEL-Centric Tool Palettes*
- *Which Edit Style to Choose: BPMN-Centric or BPEL-Centric?*

Comparing the BPMN-Centric and BPEL-Centric Tool Palettes

The following illustration shows the two palettes, followed by a discussion.
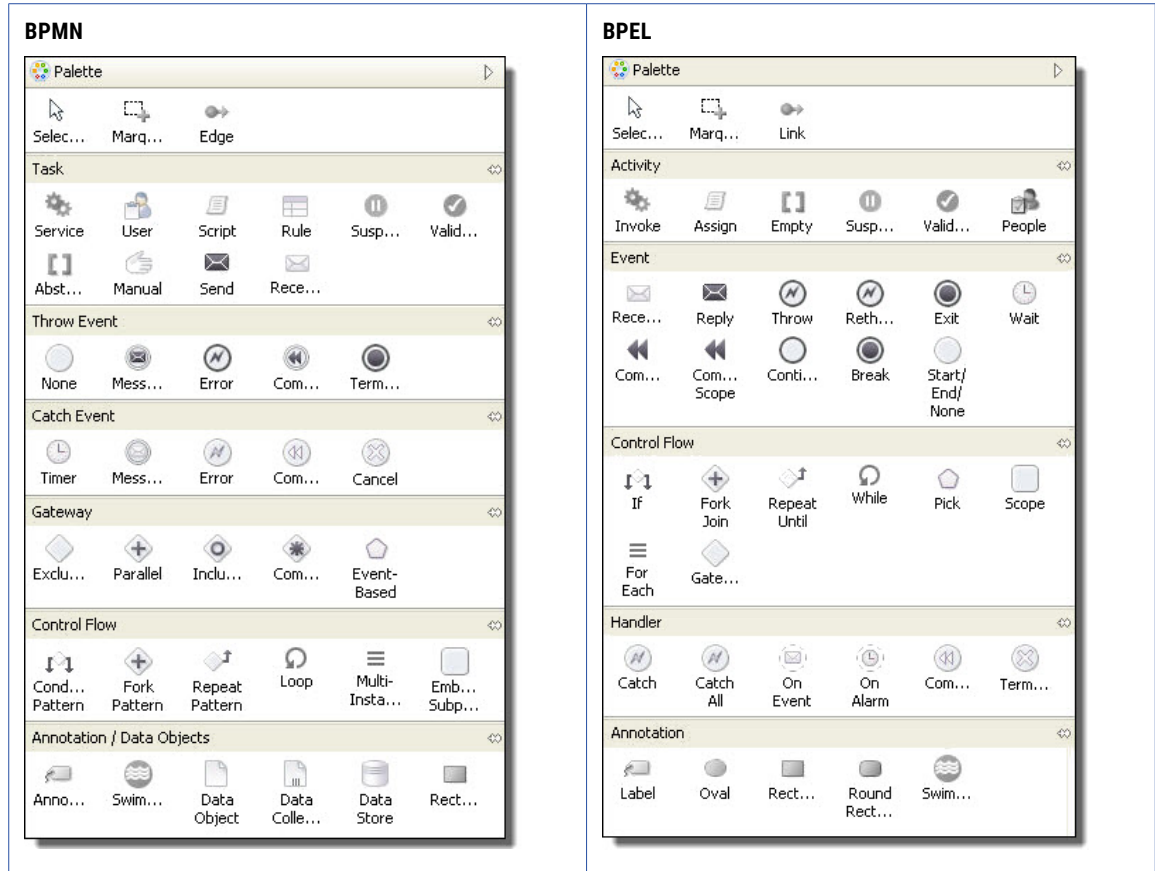
The following tables describe palette differences.

BPMN-Centric Palette

The BPMN-Centric palette is organized into the following groups:

Edges and selections	An edge connects two activities to make them run in sequence. Use the other selection tools to select a set of items on the canvas.
Task	Basic process activities to build a process. For example, select a service task to represent a Web service endpoint.
Throw Event	Events that have an impact (a result), such as a reply, invoke, throw, rethrow, compensate, exit, and break. These are outbound events.
Catch Event	Events caused by a trigger, including wait, receive, and scope handlers, which include fault, termination, event, and compensation. These are inbound events.
Gateway	Controls the divergence and convergence of sequence flows, including branching, forking, merging, and joining of paths. Includes the BPEL pick activity as an event-based gateway.
Control Flow	Containers to structure a group of activities.

Annotation/Data Objects	Add labels, shapes, and swimlanes to the canvas. They are ignored in the XML code, but they print. You can anchor a label to an activity.
Custom	Save any activity or set of activities from a BPEL drawing as a custom activity. You can reuse the custom activity in other BPEL drawings. By default, the custom palette is hidden when empty. For details, see *Creating a Custom Activity*.

BPEL-Centric Palette

The BPEL-Centric palette is organized into the following groups:

Links and selections	Use a link to connect two activities to make them run in sequence. Use the other selection tools to select a set of items on the canvas.
Activity	Basic process activities to build a process. For example, select an invoke to represent a Web service endpoint.
Event	Select an activity that triggers an event, such as a receive, to start a process.
Control Flow	Select a container or gateway to structure a group of activities.
Handlers	Select a fault or event handler for a scope.
Annotation	Add labels, shapes, and swimlanes to the canvas. They are ignored in the XML code, but they print. You can anchor a label to an activity.
Custom	Save any activity or set of activities from a BPEL drawing as a custom activity. You can reuse the custom activity in other BPEL drawings. By default, the custom palette is hidden when empty. For details, see *Creating a Custom Activity*.

See also: *Which Edit Style to Choose: BPMN-Centric or BPEL-Centric?*

Tips for using the palette:

- By default, the palette is in auto-hide mode. Rest your mouse on the palette to open it.
- Select the **Show Palette** or **Hide Palette** arrow to remove the auto-hide feature of the palette.
- Right-mouse click on a palette group title to view a list of customization options.
- Click on a palette group title to open or close the group.
- Use drawing objects to annotate your BPEL processes.
- Customize the palettes to view small icons or icons only. To view icons only, right-mouse click on any palette entry and select **Layout > Icons Only**.

Which Edit Style to Choose BPMN-Centric or BPEL-Centric

Which Edit Style to Choose: BPMN-Centric or BPEL-Centric?

Both BPMN and BPEL edit styles generate 100% validated, executable BPEL XML code. Both styles are diagramming notations for process descriptions. You can choose which visual style is easier for you and your team.

Highlights of BPMN-Centric

- The edit style uses standard BPMN terminology.

- The Process Developer `break` extension activity can be implemented with a *terminate* activity within a scope, `for Each` or `while` structured activity. The `break` implementation allows breaking out of the closest scope or loop.

- The Process Developer `continue`extension activity is not supported because no notation in BPMN has the exact Continue semantics.

- The error catch event is used for the `catch` and `catch all` fault handlers.

- The `error` throw event is used for the `throw` and `rethrow` activities.

- Annotations include icons for data objects, collections and stores.

- An annotation can be linked to an activity.

- A control flow structure, like a Conditional Pattern or Repeat Pattern, can be ungrouped into separate activities.

Highlights of BPEL-Centric

- The style uses BPEL terminology for constructs and includes some common BPMN constructs. Not all BPEL constructs are visible. Some are hidden in order to produce a cleaner, clearer diagram.

- The style includes the Process Developer extension activities break and continue

- `Catch` and `catch all` fault handlers are represented separately

- `Throw` and `rethrow` activities are represented separately

Using Swimlanes

You can provide a graphical representation of the participants in your process by adding a pool with swimlanes. The concept of a pool and swimlane is part of the BPMN specification. A pool represents a process and a swimlane represents a participant.

In Process Developer, you add swimlanes to the Process Editor canvas, typically to show each participant's activities. However, you can drag and drop activities into different lanes, using links to show the relationship between swimlane activities. There is only one pool for the process as a whole.

The following illustration shows an example of a process pool with four swimlanes, one for each process participant.
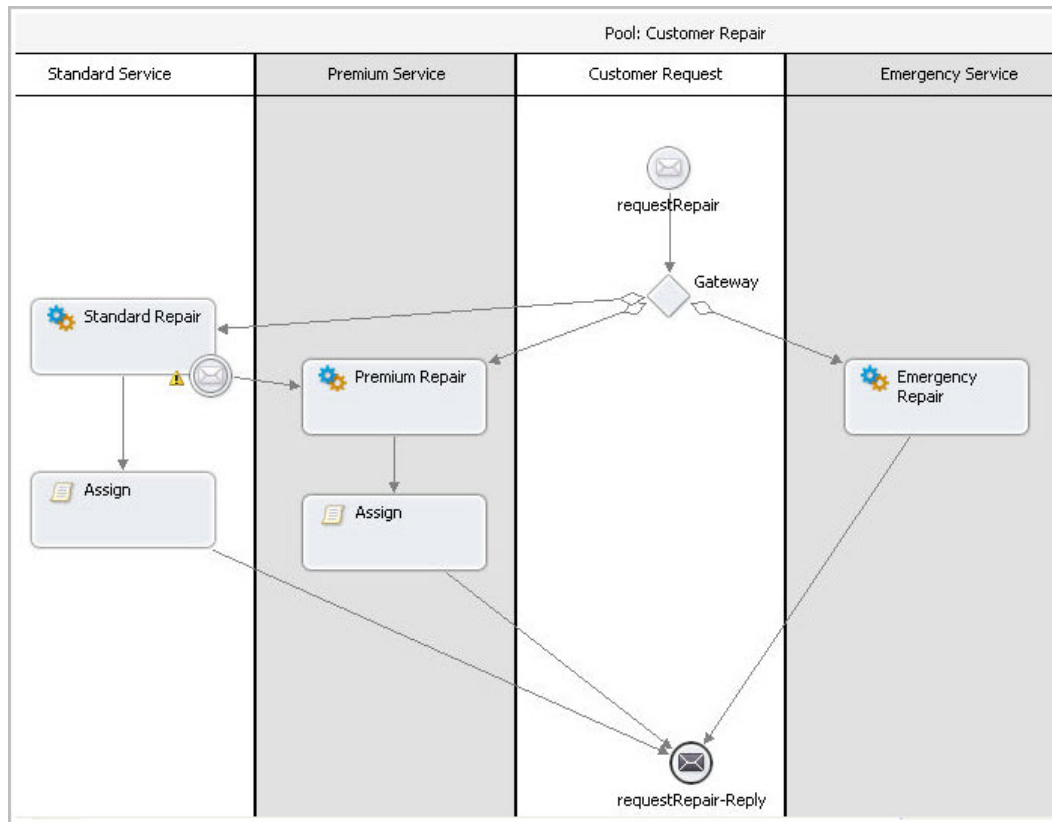
To add and use swimlanes:

1. Right-mouse click on the BPMN Editor canvas and select **Add > Annotation > Swimlane**.

2. Notice that a pool and one swimlane is created. The pool name is the process name, and the swimlane is untitled.

3. Do one of the following:

   - To add an additional swimlane to the left (or top) of the first one, right-mouse click on the left (or top) side of the process, and select **Add > Annotation > Swimlane**.

   - To add a swimlane to the right (or bottom) of the first one, right-mouse click to the right (or bottom) of the first one and select **Add > Annotation > Swimlane**.

4. To name a swimlane, select the swimlane titlebar and display Properties view. In the Text field, replace *Untitled* with your text.

If you want to change swimlane colors, you must first turn off zebra stripping.

Tips on working with swimlanes:

- By default, swimlanes are displayed in bands of white and gray, called zebra striping. To disable zebra striping and enable individual colors for each swimlane, select the title bar of the pool and open the Properties view. Set properties as desired:
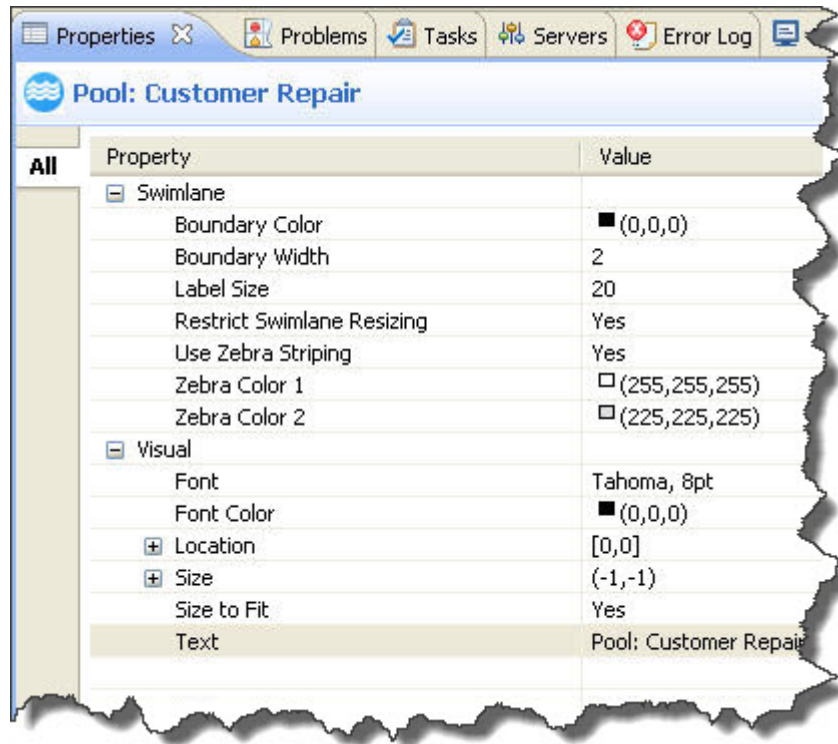
- To delete a swimlane, right-mouse click on the swimlane and select **Delete**. Delete all swimlanes to delete the pool.

- Swimlanes have only visual properties: You must manually arrange all activities within swimlanes.

- You can add a swimlane from the Annotation drawer of the palette

- Swimlanes can be used on the main canvas, not on the drill-down view of a collapsed container, nor on the fault or event handler tabs

Resizing Swimlanes

To make a swimlane grow or shrink, to accommodate the activities you add or delete, move a swimlane border either left or right in horizontal orientation (up or down in vertical orientation). By default, a resizing restriction is set on swimlanes to disallow a swimlane from being too small to accommodate an activity. The *Restrict Swimlane Resizing* property in the Pool is set to *Yes*. For flexibility in resizing a swimlane, set this property to *No*.

What is Guide Developer Classic Style

Process Developer Classic style is supported primarily for legacy processes.

Process Developer Classic uses the same palette of icons that has been in use since Version 1.0 of Process Developer. Each BPEL construct, as specified in the WS-BPEL 2.0 specification, is represented by an icon designed by Informatica. Because the BPEL specification does not include any requirements for graphical representation, each vendor who implements the specification creates their own look and feel.
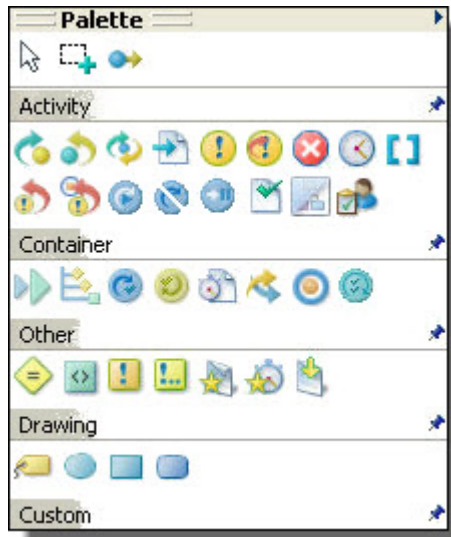
In Process Developer Classic, the palette drawers are named Activity, Container, and Other.

BPMN vs. Process Developer Classic Style Examples

The following examples show some main differences between BPMN and Process Developer Classic styles.

Example One: Shapes and Symbols

BPMN shapes have meanings. Circles indicate events, as shown with the receive and reply on the left below. Start events and end events have solid borders, while intermediate events have outlined borders. In Process Developer Classic, activities are categorized as basic and structured. Receives and replies are basic activities, represented by Informatica proprietary icons shown on the right.

Example Two: Start, End, and Exception Events

BPMN uses event icons to indicate the start and end of a workflow (in BPMN terms a *subprocess*), as shown in the scope below on the left. The start and end events are within a hidden sequence container. You add activities between start and end. When you add a catch fault handler to the scope, the fault handler is

displayed as a dotted rectangle within the scope. In Process Developer Classic, a scope starts out as an empty container. You display a fault handler with the Show Fault Handler option. Then you add a catch to it.
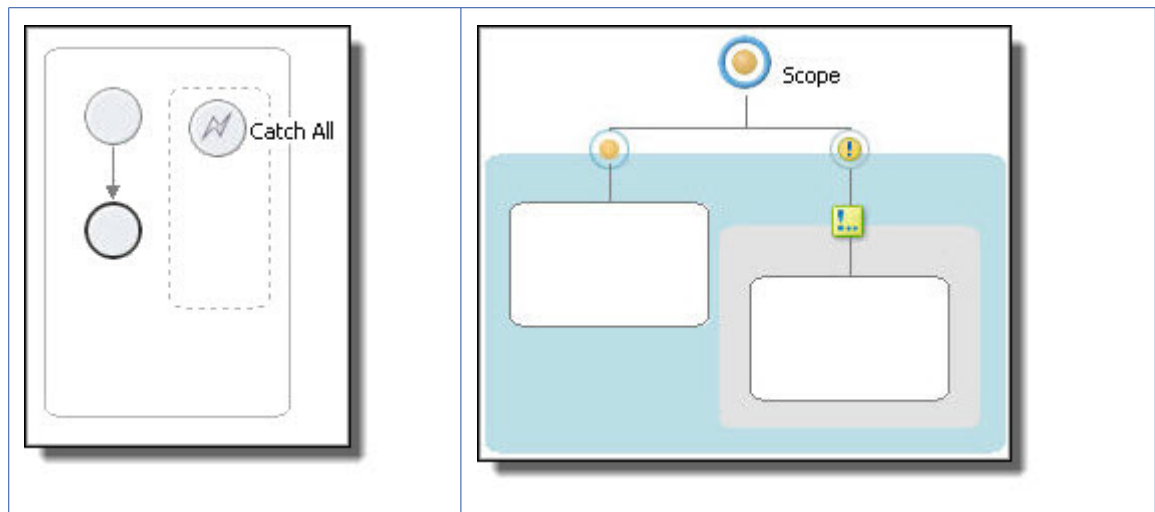
Example Three: Control Flow vs. Containers

BPEL often requires nested activities, such as those in a `forEach`, if, while, or `repeatUntil`. BPMN (on the left below) represents nesting with control flow arrows. In the `forEach` example, note also that the control flow contains an embedded (hidden) sequence. The sequence contains start and end events, and you can add activities in between. The start and end events are considered good style in BPMN, but are not strictly required, so they can be deleted if desired. Process Developer Classic displays a `forEach` container with an embedded, visible scope into which you add all activities manually.
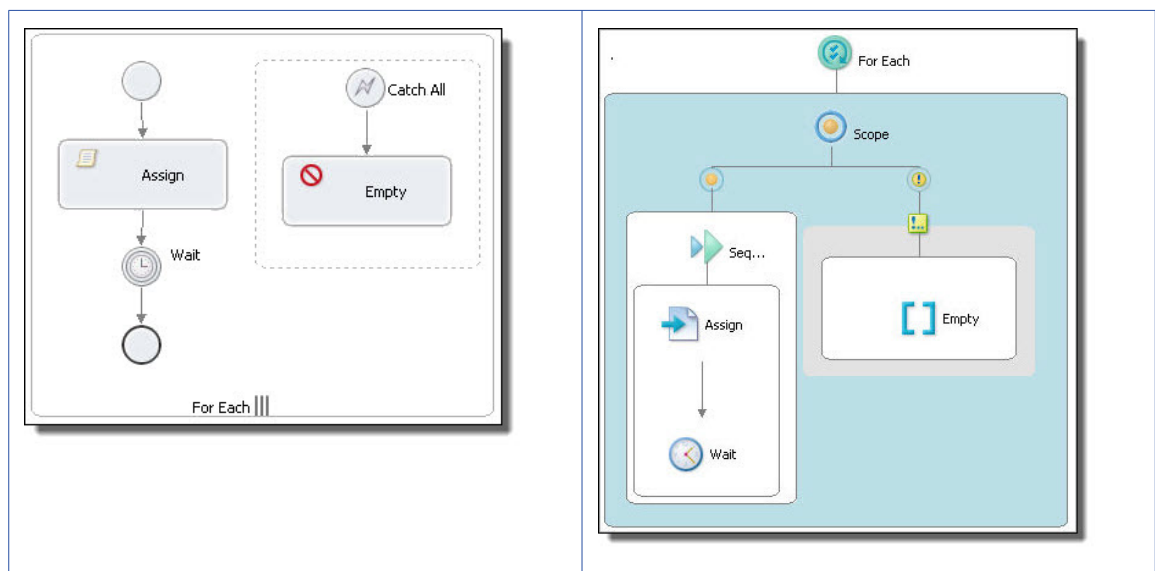
Example Four: Nested Activities

In BPMN, arrows represent control flow. Representing control flow using arrows can be easier to read than using containers, as shown in the example of nested `repeatUntil` activities. In this diagram, the parent

`repeatUntil` has two children: a repeatUntil and an invoke. The child `repeatUntil` has one child: an invoke. The BPMN diagram is easier for many people to understand.
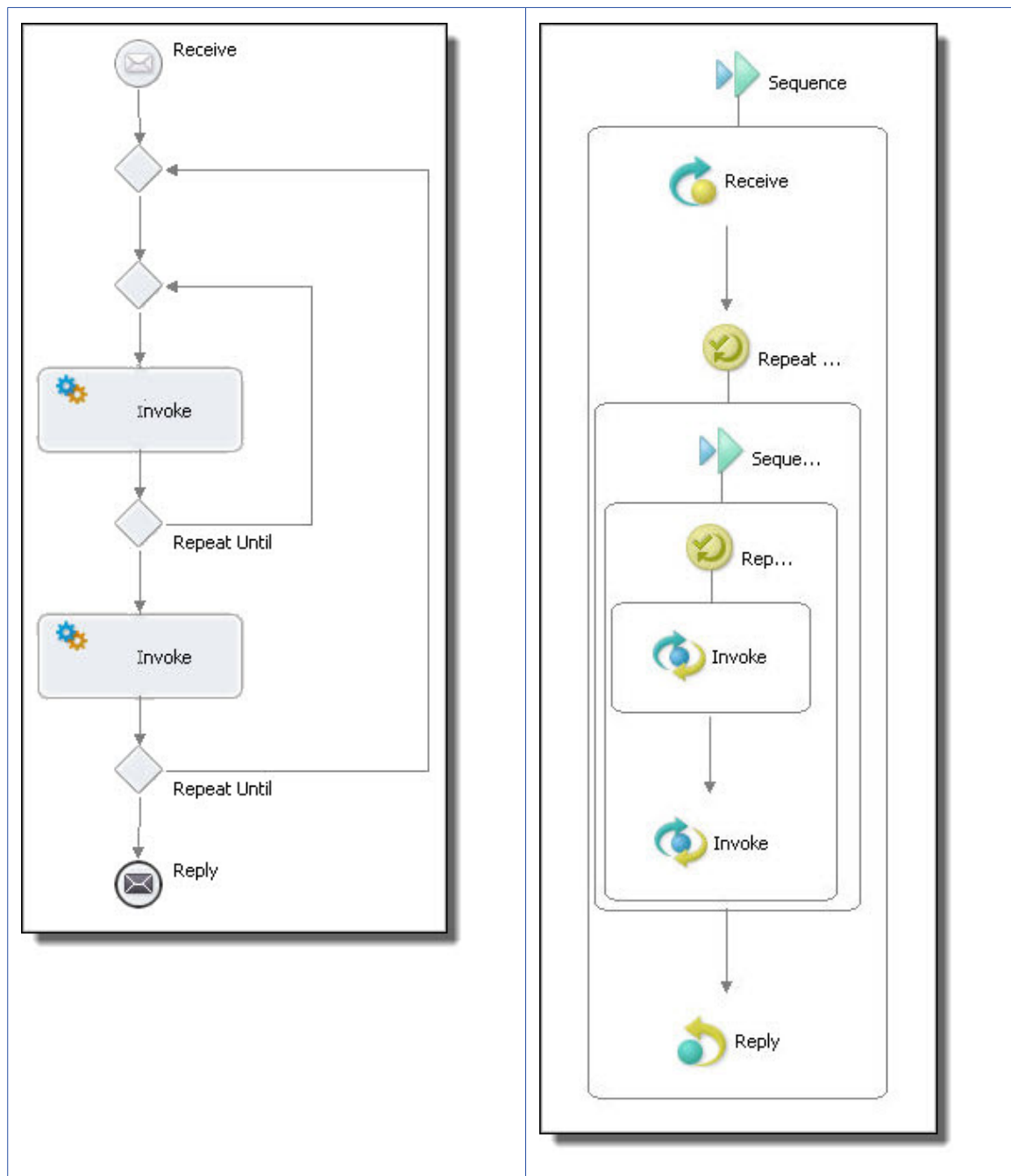
Using the Process Developer Process Editor

The Process Editor consists of a canvas and palette for creating a visual representation of a BPEL process. When you create a BPEL file, as described in *Creating a New Process*, the file opens in the Process Activities tab of the editor, and you have a canvas on which to create a process.

The following illustration shows the parts of the Process Editor.
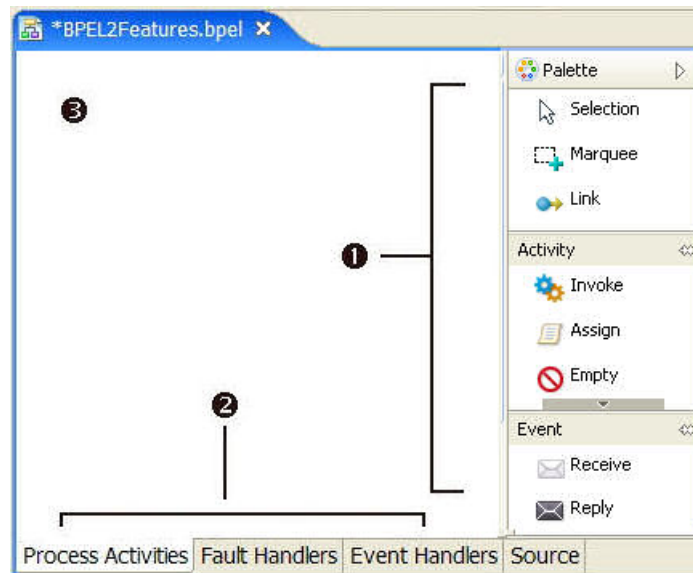
1	**Palette**. Select constructs to create a process definition. Annotate your process with drawing labels and shapes. Create custom activities to reuse in other processes.
2	**Tabs**. Select a tab to design a specific component of a process on its own page or to view the generated XML code in the *Source* tab. You can print the contents of each page.
3	**Canvas**. Drag palette items to the canvas. Set visual properties for the items and the canvas.

Process Editor Process Activities Tab

From the Project Explorer you open a process file (a `.bpel` file), and the graphical representation of the process is displayed in the Process Activities tab of the Process Editor.

Some common activities using this tab are:

- Use the Process Activities tab to design your process, including all activities, links, and structures. Select objects from the palette to design and annotate your process.

- Use the Fault Handlers and Event Handlers tabs to design handling of exceptional events at a process level.

- You can open as many `.bpel` files as you wish. Click on a file name in the Process Editor titlebar to display a specific process.

- You can print the graphical elements on this page.

See also *Tips for Designing on the Process Editor Canvas*.

Process Editor Fault Handlers Tab

Use the Fault Handlers tab of the Process Editor to design process-level fault handling. You can add `catch` and `catchAll` containers and activities on this tab. These fault handlers are added to the `<faultHandlers>` section of the BPEL file.

Typical activities when using this tab are:

- Adding scope-level fault handlers, see *Adding a Fault Handler for a Scope*.
- Adding drawing labels to fault handlers.
- Printing the graphical elements on this page.

Process Editor Event Handlers Tab

Use the Event Handlers tab of the Process Editor to design process-level message and time-out events. For more information, see *Event Handling*.

Typical activities using this tab are:

- You can add drawing labels to Event Handlers.
- You can print the graphical elements on this page.

Process Editor Compensation and Termination Handler Tabs

One BPEL process can invoke another BPEL process. In this case, the invoked process is referred to as a subprocess, and can be handled specially. The subprocess is eligible for compensation and termination handling in the same way as a process scope. For details, see *Creating a BPEL Process as a Service for Another BPEL Process*

These tabs are as follows:

- **Compensation Handler tab**. Use this tab when you enable instance compensation for the process. For more information, see *Process Element and Properties*.
- **Termination Handler tab**. Use this tab when you enable termination handling for the process. For more information, see *Process Element and Properties*.

Process Editor Source Tab

Use the Source tab of the Process Editor to view the generated XML code for your process. The code you see is validated against the WS-BPEL 2.0 specification. For details, see *BPEL XML Source and Implicitly Added Activities*.

Process Editor Source Tab

Use the Source tab of the Process Editor to view the generated XML code for your process. The code you see is validated against the WS-BPEL 2.0 specification. For details, see *BPEL XML Source and Implicitly Added Activities*.

Setting Visual Properties and Using Your Own Library of Images

For all of Process Developer's activity icons, you can change the font size, colors, borders, and other visual properties.

In addition, you can change the activity icons to impart more meaningful visual representation for the type of process you are building.

You can set defaults for all processes, as described in *Layout Preferences*, or you can modify visual properties for individual processes and activities.

Here are some examples:

Example 1: Add a border, change font size for an activity
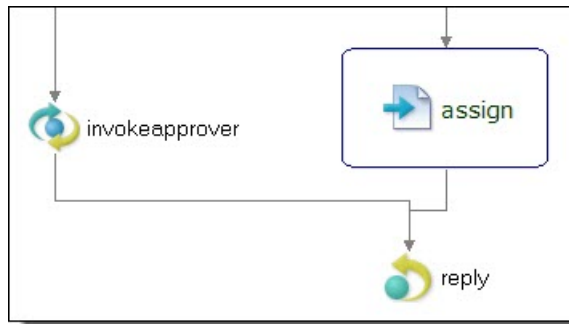
Example 2: Select your own images for activities by setting the Image Location property for each activity
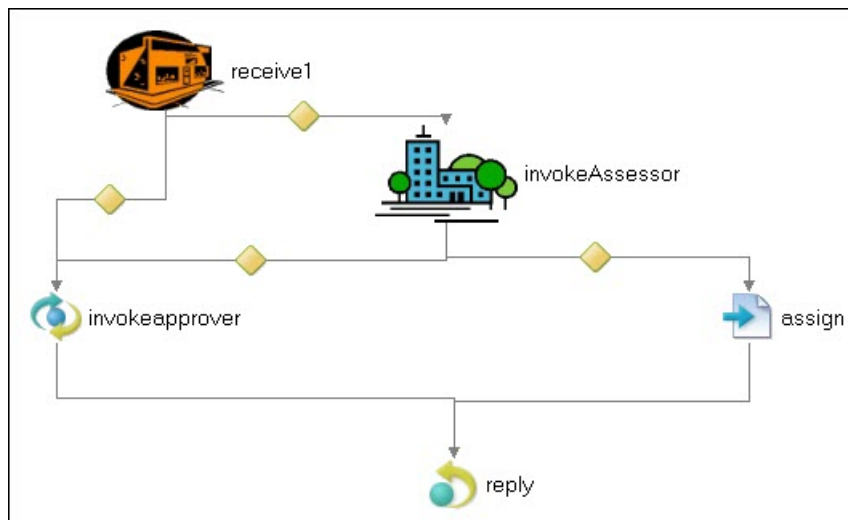
Example 3: Change the link router style from Manhattan to Manual

The manual style draws a straight link, rather than a stair-step style link, and allows you to set the angle for a link transition. A *Shortest Path* style link travels around the border of objects to link two activities.
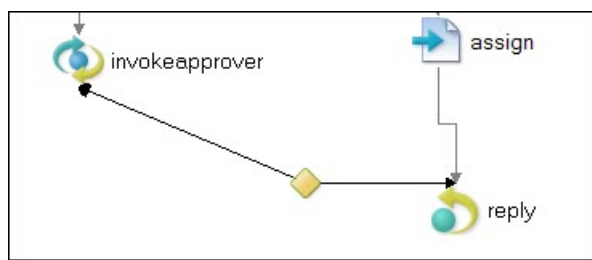
The following table lists the visual properties you can set for various activities. Not all properties apply to all activities.

Property	Description
Background Color	Fill color if the Transparent property is No
Border	Yes/No selection to display a border on a drawing item
Border Color	Opens a dialog for you to choose a basic or custom color for the border of a drawing item.
Expanded	For a container, such as a Scope, a Yes/No display indicating whether the container's contents are displayed or hidden. This setting changes when you select Expand or Collapse from the container's right-mouse menu.
Font	Sets the font to use for label.
Font Color	Sets the color for labels.
Image Display	The image to draw if any
Image Location	Sets the file system location and name of the image file. See the `Tips` section below for multi-selecting activities to change their images.
Image Text Gap	Select the size for the gap between an activity icon and the label.
Label	Select the display format for an activity label. See *Selecting Activity Labels*.
Label Alignment	Sets the align text property: left, right, center, top, bottom
Label Placement	For a drawing object, sets the placement of text label relative to image.
Location	Sets the the XY location on the canvas of the object
Orientation	For a container, sets the vertical or horizontal orientation for activities within the sequence.
Primary Alignment	For a Sequence activity, sets the alignment for the group of activities.
Secondary Alignment	For a Sequence activity, sets the alignment position of the activities within the sequence.
Size	Sets the the current size on canvas
Size to Fit	Set to *Yes* to restrict resizing of the selected drawing object. Select *No* to permit resizing of the drawing object.
Text	For a drawing object, text label
Transparency Level	For a drawing object, sets the value of transparency. A larger value decreases the transparency.
Transparent	Yes/No setting. If you select No, you can set the Background color.

Tips for setting visual properties:

- Set default values in a visual properties file. For details, see *Layout Preferences*.
- For a particular process, to change the image for a group of activities, multi-select the activities and then select the Image Location. For example, select all receives to change the icon for that activity type.

- To resize a container activity, such as an if activity, set the Size to Fit option to No.

Adding Tasks and Bookmarks to the Process

You can add a bookmark to a graphical element on the Process Editor canvas and create a link to easily jump back to that spot in the process. You can also add a task, which is similar to a bookmark. To add a task, highlight a graphical element and create a link to easily jump back to it.

A task is not related to the human interactions or people activity task element for a Human Tasks (BPEL for People) activity.

The difference between a task and bookmark is in the intent of the link. A task usually indicates something that needs to be done at that spot as opposed to just an easy link back to it. A task also has a priority to let you easily sort tasks by importance.

To add a task or bookmark to a graphical element, select the element and right-click. Select Add Bookmark or Add Task. Name the task or bookmark.

The following illustration shows a graphical element with a task or bookmark added.
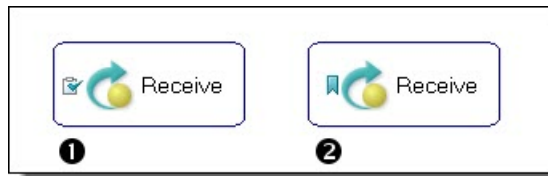
1	Receive activity with a task added
2	Receive activity with a bookmark added

For more information, see *Bookmarks View*.

Adding Comments to a Process

Process Developer offers two ways to add annotations to a process: comments and documentation.

Comments are useful for design-time annotations because you can add tasks and to do items to them. They appear as HTML-formatted comments in the BPEL source code. You can also add a documentation element to a BPEL process, activity, and links. For details, see *Adding Documentation to a Process*.

You can add a comment to any element of the process, such as an activity, link, variable, or the process itself.

1. Select an item from the Outline view or the Process Editor canvas. For example:
   - Select Process or a partner link, variable, or copy operation from the Outline view
   - Select an activity or link on the Process Editor canvas
2. In the Properties view, at the end of the *Comment* row, click **Dialog**.
3. Type in a comment in the Comment dialog and click **OK**.
   Tip: You can add a Task tag to the beginning of the comment and the comment appears in *Tasks* view. For details, see *Tasks and Problems Preferences*.

The comments for activities and links appear when your mouse hovers over them on the Process Editor canvas, as shown.

Comments appear as formatted tags in the Source view of your process.

If you want to add a comment about a namespace, add the comment to the process itself.

Adding Documentation to a Process

You can add one or more documentation items to the process or to an activity. If desired for language support, add a language identifier. The source field is for application-specific reference.

Process Developer offers two ways to add annotations to a process: Comments and Documentation. You can add a <documentation> element to a BPEL process, activities, and links. For design-time, comments are useful because you can add tasks and to do items to them. For details, see *Adding Comments to a Process*.

You can add documentation to any element of the process, such as an activity, link, variable, or the process itself.

If you generate a process report, the documentation is very useful. For details, see *Generate Process Report*.

1. Select an item from the Outline view or the Process Editor canvas. For example:
   - Select Process or a partner link, variable, or copy operation from the Outline view
   - Select an activity or link on the *Process Editor* canvas

2. In the Properties view, do one of the following:
   - Type text directly into the Documentation tab.
   - In the All view, at the end of the Documentation row, click **Dialog (..)**. Complete the dialog as shown below.

To fill in the Documentation Details:

1. Select **New** to open the **Documentation Details** dialog, as the example shows.
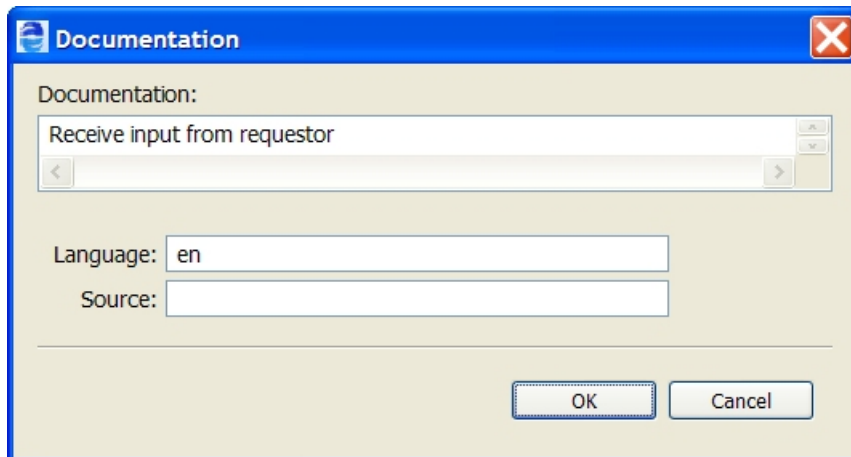
2. Fill in the dialog as follows:

- **Documentation**. Enter the content in your chosen format, such as plain text, HTML, or XHTML.

- **Language** (optional). Type in the language identifier of the natural or formal language that the content is written in. Refer to the XML 1.0 specification for language identifier reference details.

- **Source** (optional). This attribute indicates the source of application information, in the form of a URI. (The attribute is described in the XSD schema and is generally included with the documentation element.)

3. Click **OK**, and select **New** to add additional documentation elements.

4. In the Documentation entries list, you can reorganize entries by selecting the **Up** or **Down** buttons. The first entry in the list is displayed in the Properties view of the activity, link, or partner link.

Documentation appears as formatted tags in the Source view of your process and also in a report that you can generate.

Tips for Designing on the Process Editor Canvas

Use the Process Editor canvas to create a graphical representation of a BPEL process.

Here are some tips for designing your process:

- Double-click the Process Editor title bar to enlarge the canvas to full view.

- Change the font size and color, borders, images and other properties for activities and links. See *Setting Visual Properties and Using Your Own Library of Images*.

- Use the **Auto Layout** toolbar button to arrange process activities according to the settings in *Layout Preferences*. Use the *Auto Layout* option on the right mouse menu of any container or selection.

- Use Make Vertical and Make Horizontal to change the orientation of your process.

- When moving any activity or container, the object snaps to a grid. Use the Alt + mouse button to move an object one pixel at a time.

- Use **Zoom In** and **Zoom Out** on the Process menu to change magnification of the canvas.

- Select a **Zoom** percentage or **Fit Page** from the toolbar Zoom drop-down.

- Use **Expand or collapse** containers to show or hide them. Drill down into a collapsed container to temporarily view its contents. See *Showing and Hiding Activities*.

- Select two or more activities and link them using the **Link** toolbar button.

- Use the *Thumbnail View* to view different parts of your process if it is large and complex.

- Select two or more graphical elements and align them using the toolbar Align list.

- Anchor a drawing label to an activity by selecting both and then selecting **Anchor Annotation** from the right mouse menu.

Showing and Hiding Activities

You can preserve Process Editor real estate in the following ways:

- Expand and collapse container activities

- Drill down into a collapsed container to temporarily see its contents

Expanding, Collapsing, and Drilling Down into Container Activities

You can collapse the display of container and control flow activities. These activities include sequence and flow (Process Developer Classic only), fork join (BPMN only), if, while, pick, scope, repeat until, and for each.

When an activity is collapsed, select the plus icon to temporarily drill down into the activity to view its contents. Use the bread crumb display or the **Navigate > Back** command to collapse the activity.

To collapse an activity, such as a scope, select Collapse Container (or Control Flow) from the right-mouse menu. The small icon, illustrated in the collapsed scope below, indicates the container is collapsed. To expand a container, select Expand Container (or Control Flow) from the right-mouse menu.
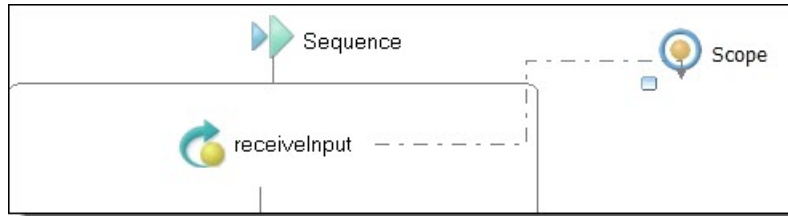
Tips for using Expand, Collapse, and Drill Down:

- When an activity is collapsed, select the plus icon to drill down into the contents. Alternately, from the right-mouse menu, select **Go Into Activity** (Ctrl + Alt + L).

- By default, the link style changes to a dotted line for a link with one or more targets in a collapsed container, as the example above shows

- You can **Expand All Containers** and **Collapse All Containers** from the Process menu or from a blank spot on the Process Editor. Right-mouse select these options from the Process Editor.

- You can set a preference to control the positioning of expanded and collapsed containers in relation to other activities. The setting affects the overall look of the process design. For details, see *Layout Preferences*.

- You can **Undo** expand/collapse moves if you do not like the results

Process Editor Keyboard Shortcuts

The following tables describe how to select and move objects on the *Process Editor* canvas without using a mouse.

Navigation Keys

- Navigate to a BPEL file in the Process Editor: CTRL + F6.

- Navigate to a selected activity with the Go to Activity shortcut: CTRL + L.

See also *Process Developer Menus and Toolbars*.

Selecting an object

When an activity or container is selected, you can navigate to another object using the following keys.

Key	Action
Space	Select
Left arrow	Navigate to the object on the left
Right arrow	Navigate to the object on the right
Up arrow	Navigate to the object above
Down arrow	Navigate to the object below

Key	Action
/ or ?	Navigate to the next link
\ or		Navigate to the previous link
Alt + Down arrow	Navigate into a container
Alt + Up arrow	Navigate out of a container

Pressing the CTRL key causes the focus, rather than the selection, to move. Pressing the SHIFT key while using one of the navigation keys extends the selection.

Moving an object

When an object is selected, you can move it using the following keys. Note that you can move an object diagonally by selecting two adjacent arrow keys simultaneously.

Key	Action
Alt + mouse	Move the selected object one pixel at a time
Period	Select next handle
>	Select previous handle
Left arrow	Drag left
Right arrow	Drag right
Up arrow	Drag up
Down arrow	Drag down
Enter	Commit the drag operation
Esc	Abort the drag operation

Palette Keyboard Actions

Key	Action
Left arrow	If focus is on an expanded palette group, such as the Activity group, then collapse it. Otherwise set focus on the group.
Right arrow	If focus is on a collapsed palette group, then expand it. If the group is expanded, then the focus moves into the group.
Up arrow	If the focus is inside a group, then it moves to the group title.
Down arrow	Moves to the next item

Process Developer Function Keys

The following function keys are available.

Location	Key	Function
Debug View	F6	Step Over
Debug View	F8	Resume

Customizing the Process Developer Perspective

As you work on various parts of your BPEL process, you may need to hide some views, enlarge others, and change the tab order of others. The workbench environment that Process Developer is built upon is very flexible and lets you do all of these things.

For details, see *Windows, Perspectives, Views, and Editors*.

Process Developer Menus and Toolbars

In addition to the Eclipse basic menus and toolbars, the following menu items and toolbar buttons are part of Process Developer:

File menu

- "Refactor" on page 70

Navigate menu

- "Go To Activity (Ctrl + l)" on page 70
- Open Declaration (F3). Goes to the WSDL or XSD file associated with the BPEL construct in focus on the canvas. For example, select an invoke and press F3 to open the WSDL and see the highlighted operation.
- "Open Operation (Ctrl + Shift + t o)" on page 71
- "Open Port Type (Ctrl + Shift + t p)" on page 71
- "Open Web Type (Ctrl + Shift + t w)" on page 71

Process menu

Items on the Process menu are associated with the process in focus on the Process Editor canvas. For details, see the following topics.

- "Using the Process Developer Process Editor" on page 59
- "Generate Process Report" on page 72
- "Generate Process Image" on page 72
- "Generate Deployment Image" on page 72
- "Validate Process" on page 72
- "Optimize Parameters" on page 71

Toolbar buttons

In addition to standard toolbar buttons, the following groups of buttons are specific to BPEL processes:

- Process Viewing aids: **Auto Layout** (see details below), **Align**, **Magnify**
- Simulation: **Start Simulation**, **Clear Execution State**
- Remote Debugging: **Process Console**, **Launch Web Services Explorer**
- Process Validation (without saving): **Validate Process**

Auto Layout Methods

Select a **Layout Method** as follows:

Method	Explanation
Grid (default)	Objects can branch both to the right/left (or top/bottom) but the layout does not optimize for available slack space. A row or column is the maximum width of any object in that row or column.
Tree	Objects can branch both to the right/left (or top/bottom) and the layout tries to use available slack space.
Hierarchy	Objects align in an outline with one main trunk and branches extend to the right

A simple process may not show any difference among the layout methods.

Refactor

Select a resource to rename it, and the files that import the resource or depend on it are also updated. For example, rename a process name and the related B-unit tests and process deployment descriptor references are automatically updated.

When you select **Refactor**, a **Rename Resource** dialog opens and you can select **Preview** to view the changes that will be made.

You can also move a resource. For example, if you move a WSDL, all other WSDL, BPEL, BPRD and PDD's are updated to point at the moved WSDL.

Rename (or move) the following resources:

- **Process name**. Note that the rename does not change the `.bpel` file name. It changes the `<process name>` element in the BPEL process.
- **WSDL file name**
- **Schema file name**
- **Schema element or type**
- **Target namespace**

In the WSDL editor, you can refactor the target namespace. This action triggers a global refactoring of namespaces for all resources that reference the WSDL or XSD. In a WSDL, if you highlight a port type, you can right-mouse click and select **Refactor > Rename** to update resources that reference this element. Similar behavior exists in the Schema editor for complex types.

Go To Activity (Ctrl + I)

Use the Go to Activity dialog to select any named activity for the BPEL process in focus.

You can select **Navigate > Go** to Activity to select any named activity for the BPEL process in focus on the Process Editor canvas. Start typing in the *Find a BPEL activity* text box to filter the activity list to matching names. The enclosing scope and process location for the selected activity are displayed.

Open Operation (Ctrl + Shift + t o)

Use the Open Operation dialog to locate an operation within a WSDL.

An *operation* is the command that is run when a Web service activity executes. A set of related operations is bundled in a port type in a WSDL. BPEL activities such as receive, reply, and invoke include operations.

You can select **Navigate > Open Operation** and use the **Open Operation** dialog to locate an operation within a WSDL. Start typing the name of an operation from a workspace WSDL to display a list of auto-completed names. The target namespace of the WSDL is displayed in the dialog. Double-click an operation to open the WSDL and highlight the operation.

Open Port Type (Ctrl + Shift + t p)

Use the Open Port Type dialog to locate a port type within a WSDL.

A *port type* in WSDL is a set of related operations. BPEL activities such as receive, reply, and invoke include operations.

You can select **Navigate > Open Port Type**, and use the **Open Port Type** dialog to locate a port type within a WSDL. Start typing the name of a port type from a workspace WSDL to display a list of auto-completed names. The target namespace of the WSDL is displayed in the dialog. Double-click a port type to open the WSDL and highlight the port type.

Open Web Type (Ctrl + Shift + t w)

You can select **Navigate > Open Web Type** and use the **Open Web Type** dialog to locate a WSDL message, schema element, or schema type. Start typing the name of any web type from a workspace WSDL or XSD to display a list of auto-completed names. The target namespace of the WSDL or XSD is displayed in the dialog. Double-click a message, element or type to open the WSDL or XSD and highlight the item.

Optimize Parameters

The **Optimize Parameters** command makes changes to your process based on the following conditions:

- The variable is only assigned within a single assign activity and possibly has an initialization expression.
- The assign immediately precedes the main activity that uses the variable as input (either in a sequence or through a link).
- The assign qualifies to be converted into an XQuery assignment type: There is only one copy expression, the copy has a from expression, which uses XQuery, and there is no initialization expression for the variable.

The assign qualifies to be converted into an XQuery assignment type if the from clause is: an expression using XPath, a literal, a variable with a property, a variable, optionally with a part or a query.

Changes made

The changes that can be made are as follows:

- If the variable is an element type, it is changed to a message type, and all copies that set it get a new "part" attribute.
- If the variable was an element type and had an initialization expression, the initialization expression is moved to become the first "copy" in the assign activity.

- If the variable is the only thing assigned in the assign statement that sets it, move the entire assign to be in the implicit scope and move all the inbound and outbound links for the assign to the implicit scope (adding to the inbound or outbound links that are already there). Otherwise, move just the copies from the assign to a new assign activity inside the implicit scope.

The actions you can perform are:

- Move the variable into the implicit scope and rename it (and all of its uses) to "parameters.".
- Convert copy "from" constructs to use expressions (unless they are from a literal). from `variable.part` with `query="/foo/bar"` becomes `$parameters.part/foo/bar`, from variable with `property="ns:foo"` becomes `abx.getVariableProperty('parameters','ns:foo')`, copy from `variable.part (query?)` to copy from an expression.

Validate Process

Select this command to validate your process using the Eclipse Validation Builder. Process validation automatically occurs when you save your process. However, for large processes, saving may take a long time. You can validate without saving with this command. Note that there is a toolbar button for **Validate Process** and a shortcut key sequence: Ctrl + Shift + V.

For details on validation, see [About Project Orchestration and Validation Builderson page 32](#).

Generate Process Report

You can use the Generate Process Report to create and export a file that summarizes the business participants and activities in a process

To generate the report:

1.   From the Process menu, select **Generate Process Report**.
2.   In the dialog, type in a filename and location where you want your file saved.

Tip: Before generating the report, add documentation to the process Properties view and to the Properties view of any process construct, especially activities. For details, see *Adding Documentation to a Process*.

Generate Process Image

Use the **Generate Process Image** command from the Process menu to create a single JPG image of the process in focus on the canvas. The file is saved to the file system location you select.

Generate Deployment Image

When you save a process, behind the scenes a screen shot of your process is automatically saved. When you deploy the project, Process Console uses this image to display the Detail Graph View of the process. The process you see on the Process Console matches that of the Process Editor canvas.

You can disable automatic saves of a process image and manually generate an image. To enable the Generate Deployment Image option, go to the process' Properties view, select the All tab, and change the Image Generation option to Manual (from Automatic). Then you must select Generate Deployment Image before you deploy if you want an exact layout match on the server.

If you do not generate the image, a default layout of the BPEL process is used that does not necessarily match the exact layout in your canvas.

The purpose of the manual option is to allow for better performance of file saves for large processes. If you experience long file saves, you can set the process' Image Generation property to Manual. Also, you may

want to disable automatic image generation for a process that is deployed as non-persistent (no process information is stored when a process terminates). Non-persistent processes are not displayed in the Process Console.

Process Developer Preferences

You can set default preferences for all processes, and then override a preference for individual processes.

To set Process Developer preferences, select **Window** > **Preferences** > **Process Developer**.

Preferences on the main page include:

Default settings for new process files:	
Target Namespace preamble	Every BPEL process needs a target namespace. Type in a default namespace preamble, if desired. You can change the namespace and add additional ones for each new process you create.
Expression Language (URI)	Process Developer supports a variety of expression languages. The default expression language is XPath 1.0. Two other expression languages are built-in. These are XQuery 1.0 and JavaScript 1.5. You can select any of these languages as the default. The default language refers to the expression language that is used (both for validation and simulation) for a given process when that process does not explicitly declare the expression language. Select **Dialog (…)** to view the configured expression languages.
Suppress join failure	Enabled by default. For a description of this setting, see *Process Element and Properties*.
Abstract process	Create all new processes as abstract, as opposed to executable, processes. Disabled by default. For a description of this setting, see *Creating an Abstract Process*.
BPEL Version	Select the BPEL specification version to use when launching Process Developer and when creating new BPEL processes. The versions supported are WS-BPEL 2.0 (current version) or BPEL4WS 1.1. The version you select sets the correct menu items, palette, and other details for new processes. Any new process you create conforms to the version chosen. Any process you open conforms to the correct BPEL version.
BPEL Extensions	
Create XPath	A Process Developer extension for WS-BPEL 2.0 processes. For a description of this setting, see *Using the Process Developer Create XPath Extension*. To use this setting during simulation of BPEL4WS 1.1 processes, see *Simulation Preferences*.
Disable Selection Failure	A Process Developer extension for WS-BPEL 2.0 processes. For a description of this setting, see *Using the Process Developer Disable Selection Failure Fault Extension*. To use this setting during simulation of BPEL4WS 1.1 processes, see *Simulation Preferences*.
Links are Transitions	A Process Developer extension for WS-BPEL 2.0 processes. Allows links that target completed activities. See *Process Developer Extension for Links*.
Add object option	

Launch primary property dialog after add	In Outline view, you can right mouse click on a node, such as Imports, Partner Links, or Variables and add a new item. If you enable this option, you can right mouse click on the item to edit the primary property.
	You can disable this option if you prefer to edit all the properties (not just the primary property) in the Properties view.
New deployment options	
Addressing Version	The default WS Addressing specification is W3C 1.0. This specification is referenced in the preamble of a new Process Deployment Descriptor file.
	You can select one of the earlier specifications if applicable for your new deployment descriptors.

These settings take effect the next time you start Process Developer. You can set additional preferences for the following:

- BUnit Preference
- Cache and Timeout Preference
- Colors and Fonts Preferences
- Contribution Preference
- Layout Preferences
- Relationships View Preferences
- Identity Chooser Preference
- Remote Debugging Preferences
- Simulation Preferences
- Tasks and Problems Preferences
- Editing HTML in Process Central Forms

B-Unit Preference

BPEL unit (B-unit) testing is a framework that tests BPEL processes in the Process Developer engine. The easiest way to create a B-unit test is to simulate a process and save the simulation as a B-unit test.

To simulate a process, you add sample data for receives and invokes.

Set the default sample data generation method for B-unit tests: inline data or imported data file. By default, the data is imported as a sample data file. This setting can make it easier for you to maintain B-unit tests. If you select inline, the data from the simulation is shown as an XML document.

In the B-unit Editor, select **Invoke** > **Simulated Response** > **Message** > **Part**, you can set the data to be inline or to be imported from the workspace.

For details, see *BPEL Unit Testing*.

Cache and Timeout Preference

You can set cache and timeout values for process validation if you experience problems when waiting for workspace builds to finish.

There are various scenarios in which WSDL and schema files are loaded, such as when opening a BPEL process that uses such WSDLs and schemas or during validation. For a process with many, large, or remote

WSDLs and schemas, load and build times can get slow. This is especially the case when you have references that do not exist or take a long time to load.

Select **Window > Preferences > Process Developer > Cache and Timeout** to view these preferences.

- **Bad WSDL Cache Timeout**. This value is set to 120 seconds. If a connection cannot be made to a WSDL URL during process validation, the WSDL location is cached so that a later attempt to connect can be made.

- **Resource Load Timeout**. This timeout controls the amount of time spent waiting to read external WSDL files. This timeout also controls how long it takes to read the WSDL (and all its imports). By default, it is set to five seconds. You can increase this value if your machine or network is slow.

For details on validation builders, see *About Project Orchestration and Validation Builders*.

Colors and Fonts Preferences

Select **Window > Preferences > Process Developer > Colors and Fonts** to view these preferences.

The Expression Builder for creating data manipulation expressions is available for several activities, transition conditions, and queries. You can set the font properties for the Process Developer Expression Text within the Expression Builder.

The BPEL Unit Testing (B-unit) view contains a toolbar button that opens a B-unit XML Compare window. You can set a preference for the text color and font that appear in this window.

For details on B-unit Testing, see *BPEL Unit Testing*.

Contribution Preference

A contribution should include all process resources to be managed as a unit of files on the server.

To view contribution preferences, select **Window > Preferences > Process Developer > Contribution**.

When you deploy a process to the Process Server, it is recommended that you include all process resources within the contributed deployment. By default, the Export Wizard selects all resources in all project folders except sample-data and test. However, you can exclude the resources if you disable **Auto-select all non-excluded project artifacts**.

You can list the project sub-folders that should be selected or excluded within Export BPR Contribution Deployment Wizard. Create a list of relevant folders to start the export wizard with.

You can also add your own folder names to the list of folders that the Export wizard should exclude. For example, if you created a backup folder, you can add it to the list.

For details, see *Managing Deployment Contributions*.

Layout Preferences

Set the default values for laying out graphical elements on the Process Editor canvas.

Select **Window > Preferences > Process Developer > Layout** to view these preferences.

The Layout preferences page allows you to do the following:

- Select an editing style defining default visual process properties.
- Define defaults for Auto Layout mode.
- Define defaults for Expand/Collapse Containers layout.

Editing Style and Stylesheets

This panel specifies the default visual properties for new BPEL processes. The default editing style is BPMN for modeling BPEL processes in the Business Process Model and Notation language. BPEL processes visually resemble BPMN diagrams.

Select *Classic* for legacy Process Developer visual representations of activities. See *What is Process Developer Classic Style?*

If you select BPMN for the editing style, you can also set *BPMN* or *BPEL*. In this panel, you can set whether you want to use the standard BPMN notation or the BPEL-centric version of BPMN. That is, the icons are mostly standard BPMN, consistent with the palette in older Process Developer versions.

For details, see *Comparing the BPMN-Centric and BPEL-Centric Tool Palettes* and *Which Edit Style to Choose: BPMN-Centric or BPEL-Centric?*.

Under Stylesheets, you can select *Other* for a stylesheet and edit the stylesheet file in a text editor to change a default value. The change applies to all processes. For example, you can set a border for the receive activity icon. In any BPEL process, you can remove the border for an individual receive activity, if desired. Note that the Classic stylesheet contains some properties that apply to only one version of BPEL, either WS-BPEL 2.0 or BPEL4WS 1.1. The BPMN stylesheet applies only to WS-BPEL 2.0.

For a description of visual properties, see *Setting Visual Properties and Using Your Own Library of Images*.

Auto Layout

The main Process Developer toolbar contains an **Auto Layout** button that organizes graphical elements on the process canvas into a grid that best fits the display area.

In this panel, set the default values for laying out the graphical elements (for example, activities and containers).

The Auto Layout Methods are available from the **Auto Layout** toolbar button. For details, see *Process Developer Menus and Toolbars*.

Make additional Auto Layout preferences as follows:

Orientation: Horizontal or vertical	Select the primary axis for your process diagrams.
Size of margins for containers	Set the size, in pixels, for interior margins for scopes, sequences, and other containers.
Spacing between levels and nodes	Set the size, in pixels, for vertical (level) and horizontal (node) spacing between activities.

Expand/Collapse Containers

In this panel, set the default values for object movement and link style of the container activities: Sequence, If, While, Pick, Flow, Scope, For Each and Repeat Until. For details, see *Showing and Hiding Activities*.

Auto Move	When a container is collapsed, objects fill in the empty space. When a container is expanded, objects move out of the way.
Auto Move Expand	When a container is expanded, objects move out of the way
Auto Layout	After a container is collapsed or expanded, objects are laid out along an invisible grid

No Move	No objects move when a container is collapsed or expanded
Collapsed Link Style	Sets the solid, dashed or dotted style for links to and from activities within a collapsed container. A style that differs from other link styles provides a visual cue for a hidden activity or container.

Activity Context Menu

By default, all relevant Eclipse context items are displayed in the right-mouse button menu of an activity on the Process Editor canvas.

To hide these items, uncheck **Show Non-Process Developer Context Menu Items.**

You can hide:

- Compare with

- Debug as

- Profile as

- Replace with

- Run as

- Team

- Validate

- Wikitext

Process Editor Tabs

You can hide the Fault Handlers, Event Handlers, and Source tabs of the Process Editor. Hide these items by removing the check mark from *Show tabs other than Process Activities*.

Relationships View Preferences

Set the default layout for Relationships view, which shows you a graphical representation of dependencies and references of BPEL, WSDL, XSD, and other artifacts in relation to the file open in the editor.

The Relationships view shows you a graphical representation of dependencies and references of BPEL, WSDL, XSD, and other artifacts in relation to the file open in the editor.

Set the preferences as follows:

- **Depth Limit**. If a file has imports and the import has imports, you can see these different levels. The default depth limit is two, for one file importing another.

- **Layout Options**. Several graphical layout styles are available. Select the one that works best for the file that is open in the editor. The default is a Directed graph, which is like a tree, but with better horizontal control.

Tasks and Problems Preferences

You can enable and create your own tags for tasks.

Select **Window > Preferences > Process Developer > Tasks and Problems** to view these preferences.

Enable user-defined task tags	Select the check box to use your own tags for creating tasks.
Add a task tag	Add a tag, and then use the tag in an activity's Comment dialog. The text is added as a Task in Tasks view. It is also part of hover help for the activity. For details, see *Add Task Tag*. Process Developer provides two default tags: @todo, with a Normal priority and TODO with a High priority.
Enable error problems	Select this check box to display all validation errors when you save your file. A validation error refers to an invalid BPEL element definition. For example, if you drag a Receive activity to the canvas without specifying the partner link, operation, and variable, error problems are generated for each missing attribute. If you do not want to be reminded of problems, deselect this check box. Before simulating execution, be sure to enable error problems and make corrections.
Enable warning problems	Select this check box to display all validation warnings when you save your file. A validation warning can refer to unused resources, such as variables and partner links, unsupported expression languages, or other potential errors. If you do not want to be reminded of warnings, deselect this check box. If your processes use Correlation Sets, it is highly recommended that you enable warning problems. A warning problem appears if correlation is missing from activities that handle correlated messages.
Enable info problems	Select this check box to display all information messages when you save your file. If you do not want to be reminded of information messages, you can deselect this check box.

Identity Chooser Preference

Select a group from your enterprise's directory. Fill in the details for the service that identifies your organization's group names. Process Developer communicates with the service to present Group Names in certain activities

Select **Window > Preferences > Process Developer > Identity Chooser** to view the settings page.

This preference, which allows you to specify the location of an identity service, is currently used only with the People activity, an activity that incorporates human interaction within the process. For details, refer to *Human Tasks* within the *Process Developer Help*.

You can set an endpoint and credential for two identity services: the Identity Chooser used for group member selection and the Admin Service used for creating a people query in the Process Deployment Descriptor Editor.

Identity Chooser

When the Process Server is running, you can open the Process Deployment Descriptor Editor, and on the People tab, there is a selection for the **Identity Chooser** dialog. When you select Refresh, this action submits a request to the Identity Service enabled in Process Server. The list of group names from your organization's directory, or from your file-based identity service, is retrieved. The group names can be selected for a Logical People Group during deployment.

On the Identity Chooser page of Process Developer Preferences, you can define communication details for accessing the identity service needed for the Identity Chooser as follows:

Endpoint	Modify the default host:port section of the Endpoint URL for the Process Server. The default URL points to a server running on localhost:8080. Modify only the host and port values to point to your server.
Username	Type in the user name credential, if needed, to access the Process Developer aeIdentityService. Your Process Server administrator can provide the user name and password associated with the abIdentityListConsumer role that enables security for the identity service from Process Developer. Information about this role is included in the *Server, Installation, Configuration, and Deployment* sections of this help.
	Note that the user name and password may be the same one you type in when deploying a BPR file as a Web Service, as described in *Creating and Deploying a Business Process Archive Contribution*.
Password	Type in the Password for the above user name

Using the Identity Chooser

Use the Identity Chooser to return a list of groups in your organization for selecting Logical People Groups in a People activity.

Before selecting the Identity Chooser, you must do the following:

- Ensure that the Process Server is running, described in *Setting Up the Embedded Process Server*.
- Ensure that the Informatica Process Identity Service Preference page is filled in correctly on the server

For details on the Identity Chooser, refer to the *Human Tasks (Process Developer) Guide*.

Admin Service

The Admin Service communicates with the server to retrieve configuration settings for creating a people query for a Logical People Group. Refer to the descriptions above for Identity Chooser for adding communication details.

Your Process Server administrator can provide the username and password associated with the abAdmin role needed for the ActiveBpelAdmin service.

Additional Preferences

In addition to the general workspace preferences described in *Process Developer Preferences*, refer to the following preference pages, described in other sections:

- *Remote Debugging Preferences*
- *Simulation Preferences*

Accessing Process Developer Help

Process Developer has many online help features:

Help Feature	Description
Help menu	Select the Help menu to view the Welcome page, Tips and Tricks, and Help Contents
Welcome page	Provides getting started help for new users
Help Contents	From the Help menu, displays all help topics in an expandable/collapsible list
Cheat sheets	From the Help menu, opens a view containing common, self-guided, getting-started tasks in a step-by-step format
F1 Help	Press F1 on any view or dialog box to view a context-sensitive help topic
Hover Help	Rest your mouse on an item in a list, toolbar button, or graphical display to view property details
Help Search	Search for help topics by selecting the Search menu or toolbar icon

CHAPTER 4

About Interfaces Service References and Local WSDL

Every BPEL process requires an interface for Web interaction activities, such as receives and invokes. The interface is a Web Services Description Language (WSDL) file that describes the services and messages to be exchanged in an orchestration. Process Developer provides easy ways for you to add, import, create, or use built-in WSDL files, namely through:

- Importing a local WSDL file into your project.
- Importing a reference to a remote WSDL.
- Creating a new WSDL file based on a new participant.
- Creating a new WSDL file based on sample data, schema, or Java interface.
- Using a system service.

When you add a new WSDL file to an orchestration project, it is available to all processes.

By adding WSDL files and schemas, you have a convenient registry of namespaces, messages, type definitions, sample data, and other elements to use across BPEL processes. To make your work quick and easy, wizards help you jump-start process design by automatically including the WSDL elements needed.

In addition, when you add a WSDL, you can add sample data files for message variables and make the data available for testing and debugging your process.

To display WSDL interfaces, you can select Participants or Interfaces view.

WSDL files are XML-based files that contain the namespace declarations, type declarations, messages, and other element definitions that you can use to create business processes. WSDL files conform to the Web Services Description Language standard authored by a World Wide Web Consortium committee.

Process Developer supports Web Services Description Language (WSDL) 1.1, a W3C Note dated 15 March 2001. For more information about this standard, see `http://www.w3.org/TR/wsdl`.

Importing a Local WSDL

As described in *Creating an Orchestration Project*, you can add folders for managing process resources to your project in the Project Explorer. One of these folders is for WSDL. You can import WSDL to this folder, or anywhere in the Project Explorer, to have a locally available WSDL for use in your processes.

Once you import a WSDL and its schema into Project Explorer, the port types are added to Participants and Interfaces views.

To import a WSDL:

1. In Project Explorer, right-mouse click on a `wsdl` folder (or select a project).
   If the WSDL imports a schema or other WSDLs, import the files first to avoid seeing validation warning messages.

2. Select **Import**.

3. In the **Import** dialog, select File System for the Import source.

4. Browse to your WSDL and select it.

5. Note that the WSDL port types are added to the Participants view.

You can manage WSDL files as follows:

- *Viewing Key Elements of a WSDL Tree*

- *Deleting a WSDL from Your Project*

Viewing Key Elements of a WSDL Tree

The Project Explorer displays the names of imported WSDL files. Because a WSDL file is an XML file, you can expand or collapse the file view in a tree structure to see elements.

To display the WSDL file as an XML tree structure, click the plus sign next to the file name to show the elements in the file:

- Bindings and operations

- Messages

- Partner link types, roles, and operations

- Port types and operations

- Properties and property aliases

- Service names and ports

Editing a WSDL in the WSDL Editor

You can double-click a WSDL file in the Project Explorer to view and edit it in the WSDL Editor. Double-click an operation or port type to highlight it in the editor.

Your WSDL changes can affect the references in BPEL files, and you may need to resolve them. For example, if your BPEL process refers to a partner link role name that you change, the role name is marked as Unresolved Reference, and you must select a new name from the Properties view to ensure your process is valid.

Deleting a WSDL from Your Project

You can delete a WSDL you no longer need. Select the WSDL file from the Project Explorer, right-mouse click, and select **Delete**. Any processes using the WSDL become invalid, since the project location is no longer valid. The partner link type and port type entries are removed from the Interfaces view.

Importing a Service Reference

Import a remote WSDL from your server or repository. Keep your local copy in synch with it.

A *service reference* is a project-based catalog of external WSDL services. You can import external services to a project, while still maintaining a connection to the external location. A service reference is treated as the original location, but resolved locally.

Using a service reference lets you maintain your local copy of WSDL and schema while maintaining an awareness of changes being made to the original files. This technique allows you to decide how best to handle updates within your local copy.

By importing a service reference, you can:

- Compare the local WSDL file with the original source.
- Update from the original source by manually editing your local copy if the changes do not adversely affect the processes that import the WSDL file.
- Use the WSDL just as you use a WSDL imported into the WSDL folder.

To import a locally-connected copy of an external WSDL:

1. Right-mouse click on the `Service Reference` folder in your orchestration project and select **Import**.
2. Paste in the URL of a WSDL file that is on your network server, intranet, or Internet.
3. Associate imported services with a `Service Group` folder. A `Service Group` folder contains the imported service and all of its imports, such as WSDL and XSD files. The suggested service group name indicates the location and name of the original WSDL. When you select **Finish** on the **Import** dialog, you will see the service group within the `Service References` folder.
4. Enable the checkbox for Format file after import to make the XML more readable. The import wizard tries to indent child elements and create line breaks.

To compare a Service Reference with its Original Source:

To compare a local WSDL file with its original source, right-mouse click on the `Service Group` folder and select **Compare with > Import Source**. If the files are not in sync, a WSDL file appears in a *Compare* window. Click on the file to view the differences between the local and the original copies. You can only view the differences. Editing is not allowed.

Tips on Using Service References:

- If the original file is updated with new or changed port types or schema changes, and you edit your local copy to match the original, a process importing the WSDL will contain errors
- If the original file becomes unavailable, you should copy the WSDL or schema into your project and then change the import location to point to the project location. This avoids the potentially long waits for remote validation to occur. You change the import location in the Properties view of the import. Imports are listed in the Outline view.
- For details on setting timeout values for WSDL validation, see *Cache and Timeout Preference*.

Creating a New Interface

You can use the New Interface wizard as a quick start to generate a basic WSDL file if you do not already have one.

If you do not have a WSDL file, you can use the New Interface wizard to generating the basics of a WSDL file. The starting point for generating a WSDL file is to have one or more of the following as a basis for input and output messages of a port type and operation:

- XML schema
- Sample XML data
- Java interface

To create a new interface:

1. Do one of the following:

   - Add an XSD or XML file to a project folder within an orchestration project. These files should describe data types, and must include a namespace definition.
   - Follow the steps described in *Creating a Java Interface*.

2. Display the New Interface wizard, either by creating an interface for a participant (Participants view) or by selecting the **New Interface** icon from the Interfaces view toolbar.

3. In the New Interface wizard, select whether the new operation is synchronous or asynchronous:

   - Request-response operation
   - One-way operation

4. Select the data type you want to use to describe each of the operation's messages:

   - XML schema
   - Sample XML

5. Browse to the schema or sample and select it. For a schema, select the element to use for the message. If the wizard reports errors, verify that your data file contains valid definitions, declarations, and syntax. For examples, see the sample data files and schema that are in the tutorial project.

6. View the message in the preview.

7. Based on the names you selected for input and output, the wizard creates the required WSDL elements. You should change the Port Type, Operation, and Target Namespace elements to more meaningful names.
   The target namespace is a convention of XML Schema that enables the WSDL document to refer to itself. It is a value that is unique, and is different from all other namespaces that are defined.

8. Save the new WSDL file with the suggested name, or type in a new name and save the file to a project location.

9. View the generated WSDL in the WSDL editor to make corrections and additions.

The generated WSDL is a good starting point for a process interface. You can to modify the WSDL as follows:

- Correct or add a schema location.
- Add fault names and messages.
- Add another operation and add to the participant (or partner link type) generated. (A partner link type can have two roles. They represent two different services that communicate asynchronously.) Alternatively, in the Participants view add a callback interface. See *Creating a New Callback Interface*.
- Replace the `tns` prefix with a meaningful one. When you use the operation wizard to create activities, the prefix is automatically added to the process.

Creating a Java Interface

Process Developer provides an easy way for Java developers to use existing Java projects to build Java endpoints in a BPEL process. Your project can include a POJO (plain old Java object) or EJB. (A JavaBean can be used for all application servers except for Apache Tomcat). You can either start with or build a package that includes an interface and a Java class that implements the interface.

Using one of the techniques that Process Developer provides, you can automatically generate a WSDL and schema from your Java interface. The WSDL includes the port type, operation, and messages to create receives, replies, and invokes.

Process Developer also provides built-in features to automatically include all the JAR and other files required for deployment to the server.

Stateless or Stateful Invokes

Using the Java Interface, you can easily create invoke activities in BPEL. These invokes can be stateless or stateful.

- A stateless invoke means that each time the Java code is called from the BPEL process, a new instance of the Java class is instantiated.
- A stateful invoke is based on a Java class that is marked implementing the `java.io.Serializable` marker interface. With a stateful invoke, the process uses the same Java instance throughout the lifetime of the partnerLink useing it. Multiple invokes on the same POJO partner link result in method calls on the same POJO instance.

Setting up Your Java Project in Process Developer

To create a Java interface for a BPEL process, you can start with an existing Java project or create a new Java-enabled orchestration project.

Setup a Java project a Java Project:

1. Select **File > New > Orchestration Project**.
2. Name your project and click **Next**.
3. Select Java Enabled Orchestration.
4. In the orchestration project, import or create a new Java Project. The Java project must contain:
   - A Java interface with at least one method.
   - Concrete implementation of the Java interface.

Tip: If you have a Java project open in Eclipse, you can switch perspectives from the Eclipse Java Perspective to Process Developer. Then in the Project Explorer, right-mouse click on the project, and select **Add Orchestration Nature**.

When you add an orchestration nature to your Java project, a background validation builder ensures that you are aware of errors and problems in your process, imported WSDLs, and its other resources. Also, the builders automatically synchronize the generated WSDL with changes that might be made to the source Java.

After adding the Java project to Process Developer, you can generate a WSDL that is the BPEL-oriented interface needed to create activities. For details, see *Generating WSDL and Schema from a Java Interface*.

See also:

- *Constraints for your Java Project*
- *Creating a Java Interface*

Constraints for your Java Project

The following requirements exist for Java classes used in a Java Interface implementation:

- The Java interface's method parameter types are limited to JavaBeans and the following primitive Java types:
  - byte, short, int, long, float, double, boolean, char
  - Byte, Short, Integer, Long, Float, Double, Boolean, Character
  - String
  - BigInteger
  - Date

- Java checked exceptions are modeled as faults, but the Java method must throw the same exception at runtime for the fault to be properly thrown.

- The Java interface to the WSDL generator is performed by JAXB, and support for JavaBeans is inherited from JAXB.

- Arrays and Collections are not currently supported as arguments to Java methods; however, Collections are suwpported if they are wrapped in JavaBeans.

- The class that implements the interface must have a default constructor.

- The class that implements the interface can be stateless or stateful. A stateful class is one that is marked implementing the `java.io.Serializable` marker interface.

See also *Creating a Java Interface*.

Generating WSDL and Schema from a Java Interface

No WSDL? You can use the New Interface wizard as a quick start to generating the basics of a WSDL.

Generating WSDL and Schema from a Java Interface

Once you have set up your Java project in Process Developer's Project Explorer, you can generate WSDL and schema. A BPEL process requires a WSDL interface, and Process Developer automatically generates it from your Java interface.

Additionally, Process Developer automatically synchronizes the generated WSDL with any changes that you might make to the source Java.

To generate WSDL and schema:

1. In your Java orchestration project, create a new BPEL process in the bpel folder.

2. In Participants view, right-mouse click on Partner Service Provider and select **New Partner Service Provider**.

3. In the **Partner Service Provider** dialog, select **Generate Interface**.

4. In the New Interface wizard, select From a Java Interface.

5. Browse to your Java interface and select it.

6. Notice that the new WSDL is generated in the wsdl folder of your project.

7. Select **Finish**, and then select **OK** in the **Partner Service Provider** dialog.

WSDL Generation

The following actions occur when you generate a WSDL and schema from a Java interface:

- All methods in the Java interface manifest as WSDL operations.

- All method arguments manifest as top-level XML schema element declarations. The arguments are generated with generic names, such as "argl_string". To generate actual argument names, see *Generating Argument Names for Schema Elements*.

- The method return type manifests as a top-level XML schema element declaration.

- All declared exceptions on the method manifest as WSDL faults.

- The WSDL is commented with "Do Not Edit" since it is a derived file that can be automatically updated.

The resulting WSDL contains the following constructs:

- One Partner Link Type named [*JavaInterfaceName*]PLT

- One Port Type named [*JavaInterfaceName*]

- One Operation within the port type for each method in the Java Interface

- Two messages for each Java interface method:

  - One message for the method's arguments named [*methodName*] with a single part of the same name.

  - One message for the method's return value named [*methodName*]Response with a single part of the same name.

- Two embedded schema elements for each Java interface method:

  - One element to model the Java interface method's arguments named *methodName*.

  - One element to model the Java interface method's return value named [*methodName*]Response.

- A schema complexType and a schema element for each Java class used by the interface.

Schema Generation

- Top-level element declarations are created for each of the explicit class references.

- The implicit declarations manifest as top-level complexTypes and are reused throughout other parts of the schema (if applicable).

More Ways to Generate WSDL and schema:

Technique to Generate WSDL	Result
In the Interfaces View, select the New Interface toolbar button.	The New Interface Wizard opens for you to select a Java Interface
In the Project Explorer, select the interface from your Java project. From the right-mouse menu, select **Generate WSDL**	WSDL and schema are generated in the location you select
From your Java project, drag and drop a Java interface method to the Process Editor canvas	The Create Activity wizard opens so that you not only generate WSDL and schema, you also create a new Web Services activity (such as a receive).

See also:

- *Updating Your Java Project and Your BPEL Process Concurrently*

- *Creating a Java Interface*

Generating Argument Names for Schema Elements

When you generate a WSDL file, as described in *Generating WSDL and Schema from a Java Interface*, a method's argument are generated as schema elements. The elements have generic names, such as `arg1_string`, as shown:

```
<xs:element form="unqualified" name="arg1_string" type="xs:string"/>
```

You can generate schema elements with the actual argument names by adding a JAR file called `ae_rt.jar` to the project's Java build path list of external jars and adding annotations to your interface prior to generating the WSDL.

To generate argument names:

1. In the Project Explorer, right-mouse click on your project, and select **Properties > Java Build Path**.

2. Select the Libraries tab.

3. Select *Add External JARs*.

4. Browse to [*Designer Installation*]`\designer\plugins\org.activebpel.enginep_`[*version_number*]`\server\shared\lib`

5. Select `ae_rt.jar`, as shown:
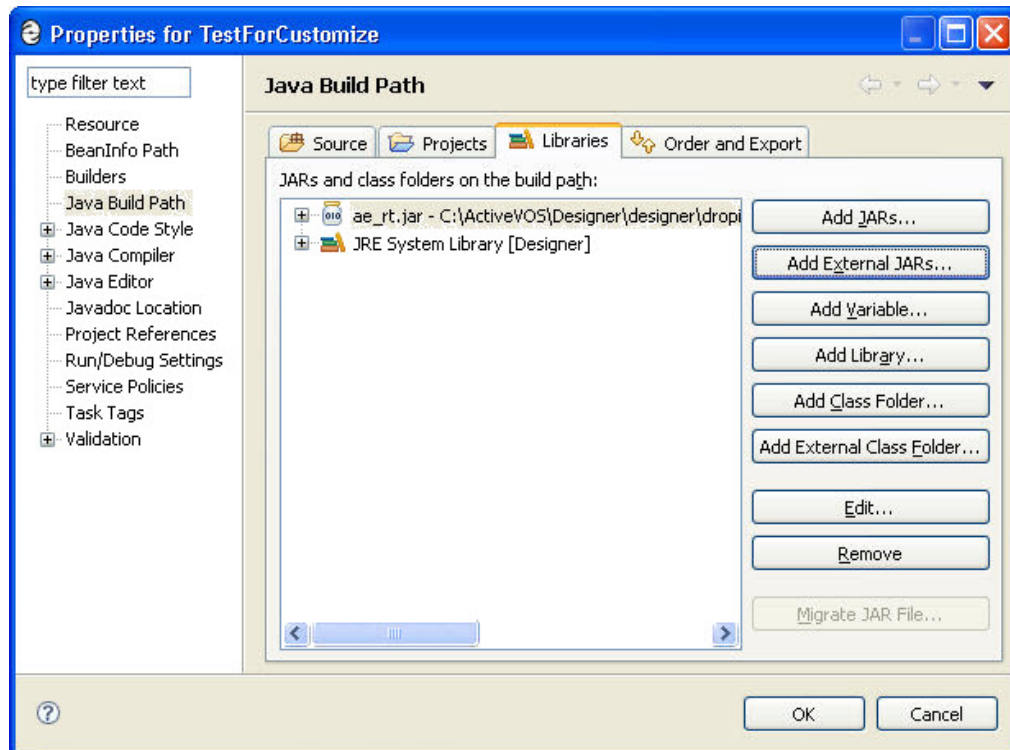
6. Open your Java interface file and add the `@AeWsdlParam` annotation to each argument. For example:

```
Public interface IMyInterface {
        public void execute(
          @AeWsdlParam("someValue") String someValue);
}
```

7. Generate (or regenerate) a WSDL by right-mouse clicking on a Java interface, and selecting **Generate WSDL**.

Process Developer generates the schema element with the argument name; for example:

```
<xs:element form="unqualified" name="someValue" type="xs:string"/>
```

Notes:

- You should not edit the generated WSDL. If you update the Java interface, you must regenerate the WSDL.

- You do not need to deploy `ae_rt.jar`. It is already on the server's `classpath`.

- If you upgrade Process Developer to a new version, you must remove the older version of `ae_rt.jar` and re-add the new one.

Updating Your Java Project and Your BPEL Process Concurrently

You can actively develop both the Java code that is being invoked and the BPEL that is invoking it. Process Developer regenerates the WSDL and schemas whenever the source Java interface or classes change. For example, if you add a new method to the Java interface, Process Developer automatically regenerates the generated WSDL and all generated schemas.

If the WSDL changes, you will see unresolved WSDL references indicators in your BPEL process. Error messages indicate the fixes you need to make for each activity using the WSDL.

See also *Creating a Java Interface*.

Deployment Requirements for a Java (POJO) Endpoint

Select the Java class that will be created and invoked at runtime.

For details on EJB Service, see *Deployment Requirements for Java (EJB) Endpoint*.

After you develop a BPEL process with a WSDL based on a Java interface, you're ready to deploy by creating JAR files. You will then provide deployment details specific to the Java interface. When you create the Process Deployment Descriptor (PDD) file for the process, you must provide details for the Java class that is created and invoked at runtime.

To provide required POJO deployment details:

1. Open the PDD Editor, as described in *Creating a Process Deployment Descriptor File*.

2. On the Partner Links tab, select the partner role that invokes the Java interface.
   Notice that the selection is set to Java Service for the invoke handler.

3. In the **Java Invoke Handler Properties** dialog, select the Java class that is created and invoked at runtime.

4. Select the Java classpath to be used when instantiating and invoking the Java class. The classpath includes not only the JAR (or local "src" workspace directory) that contains the class being invoked, but also any JARs that contain classes referenced directly or indirectly by that class.
   Note that the Inherit server classloader checkbox is selected by default. If you disable this option, you will only have access to the JARs on your classpath, not the JARs that are deployed to the server. You can also select Parent classloader last when inheriting Process Server classloader.

Deployment Details

When you select a Java invoke handler for the partner role implementing the Java interface, the Java resources are added to the resource catalog and deployed with the process. This includes all of the JARs included in the classpath.

Also, there is no endpoint reference information needed for a Java service.

See also *Creating a Java Interface*.

Deployment Requirements for Java (EJB) Endpoint

Select the Java class that will be created and invoked at runtime.

For details on setting up and designing a BPEL process with an EJB service, see *Creating a Java Interface*.

If you have developed a Web service implemented as an EJB, you can deploy the JAR files and other resources using the partner role invoke handler called EJB Service. Note that in the PDD Editor, the invoke handler detects your EJB project as a POJO and automatically selects Java Service. Be sure to select EJB Service.

Also note that an EJB Service cannot be used on Apache Tomcat. It is supported on other application servers.

The EJB Service handler performs a JNDI lookup to obtain the EJB home to instantiate the object used to call the method.

Note: Effective in version 9.2.6, support for WebLogic and WebSphere in ActiveVOS is deferred and is unavailable.

Deferred functionality is either not available or not supported in the current release. Informatica intends to reinstate it in an upcoming release, but might choose not to in accordance with changing market or technical circumstances.

To fill in the EJB Service Invoke Handler Properties Dialog:

1. For a JNDI Name, specify the JNDI name as defined in your application server's EJB file, such as the JBOSS ejb-jar.xml file, the WebLogic weblogic-ejb-jar.xml file, or the WebSphere ibm-ejb-jar-bnd.xml file.

2. For EJB Home, specify the home interface as defined in application server's ejb-jar.xml file.
   This property is required for WebSphere, but is optional for JBoss and WebLogic.

3. In the Classpath box, select the Java classpath that is used when instantiating and invoking the Java class. The classpath includes not only the JAR (or local "src" workspace directory) that contains the class being invoked, but also any JARs that contain classes referenced directly or indirectly by that class.
   Note that the Inherit server classloader checkbox is selected by default. If you disable this option, you will only have access to the JARs on your classpath, not the JARs that are deployed to the server.

When you select an EJB invoke handler for the partner role implementing the EJB interface, the Java resources are added to the resource catalog and deployed with the process. This includes all of the JARs included in the classpath.

Also, there is no endpoint reference information needed for an EJB service.

Running a BPEL Process with a POJO EJB Endpoint

At runtime, when an invoke activity based on a Java endpoint executes, Process Developer invokes the Java method as follows:

- Unmarshalls the WSDL message into Java objects. (For a POJO, reflection is used. For an EJB, a JNDI lookup is used.)

- Instantiates the concrete Java class.

- Invokes the appropriate method, based on the WSDL operation name.

- Marshalls the method return value into a WSDL message (or marshalls the Java exception into a WSDL fault).

Comparing the POJO EJB Interface to a Custom Java Invoke Handler

The Java Interface is designed for ease-of-use. A WSDL is generated for you and the JARs and other resources are automatically deployed in a BPR file. However, you must keep the Java project simple by following the requirements outlined in *Constraints for your Java Project*.

If your BPEL process requires custom Java code that handles asynchronous call-backs, policy assertions, and other details, you can use the Java wrapper custom invoke handler, as described in *EJB/Java Invoke Handler Properties Dialog*.

Using the Interfaces View to Create Activities

For an easy starting point for creating activities, see *Participants*, where you can use options instead of using Interfaces view. If you are not familiar with BPEL requirements for partner link types, you can begin in the Participants view.

One of the key productivity features of Interfaces view is accessing WSDL operations for use in your process designs. You can begin by creating an orchestration project, import your WSDL, and then from Interfaces, drag a port type operation to the Process Editor canvas. For details, see *Creating an Activity by Starting with a WSDL Interface*.

- If no WSDL elements exist in Interfaces view for an activity you need to create, select **New Interface** from the Interfaces toolbar. A wizard appears that helps you create a new WSDL. For details, see *Creating a New Interface*.

- For system service activities, such as sending an email or retrieving a user from an identity service, use an operation from a built-in WSDL. For details, see *System Services Interfaces*.

- To use a Java interface, see *Creating a Java Interface*

Interfaces Toolbar Options

Use the toolbar options to add a new interface or filter interfaces to show only the ones you are currently interested in.
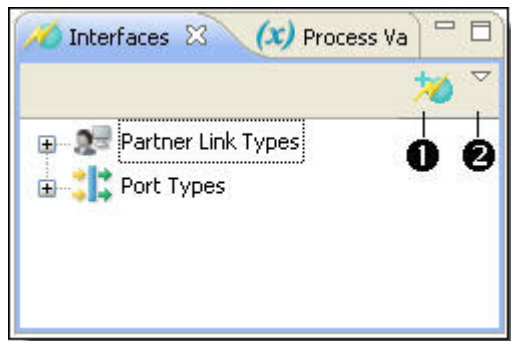
1	*Creating a New Interface*
2	*Filtering the Interfaces View*

Filtering the Interfaces View

Create or edit a Reference Set. Also, select a Reference Set to filter Web Interfaces view.

By default, Process Developer shows all interfaces for all projects in the workspace. If you have several BPEL files open at the same time, you can filter the Interfaces view to display only the interfaces relevant to the BPEL file in focus or the project in focus.

You can use a filter in the following places:

- **Workspace**. All interfaces.
- **Project**. Interfaces from the currently open project.
- **Process**. Interfaces from the BPEL file in focus.
- **System Services**. Interfaces for invoke activities.

System Services Interfaces

When you select System Services in the new **Participant** dialog or in the filter in *Interfaces* view, you see interfaces that are defined within Process Developer WSDLs. These WSDLs are exposed so that you can create activities that can communicate with the Process Server.

The following is a description of system services you can use in receives, replies, invokes, and People activities.

The following is a description of system services you can use in receives, replies, and invokes.

Partner Link Type/Operation	Description
Process Consumer Services	
Alert Service	
Event Action	
Monitoring Alert Service	
REST Service	For details, see Using a REST-based Service
Retry Invoked Servic	For details, see
Guide Designer Service Call Step Service	For details, see the *Process Developer Guide Designer* documentation.
Task Custom Notification	Creates a custom process to execute for a People activity's notification deadline. For details, see *Process Developer Human Tasks Help*.
Partner Service Provider Services	
Process Developer Admin Operations	The operations provide an API to Process Developer administration functions
REST Service	For details, see Using a REST-based Service
Send Email	For details, see Email Service

Partner Link Type/Operation	Description
Server Log Commen	
Shell Command Execute	

Creating a WSDL File with Extensions for BPEL

Not all WSDL files include the extensions required to create executable BPEL processes. The extensions are partner link types, properties, and property aliases.

Tip: Use the Participants view for automatic creation of partner link types.

You can use a Process Developer wizard to create partner link type definitions that are required for creating executable BPEL processes.

When you create new partner link types, Process Developer asks if you want to add them to an existing WSDL or create a new WSDL. If you create a new WSDL, it is automatically added to the project and catalogued in Interfaces. The WSDL contains only the partner link type definitions.

In addition, a wizard helps you create property and property aliases to be used for correlation sets. These extensions can be added to an existing WSDL or to a new WSDL.

For details, see *Partner Link Type* and *Adding Variable Properties and Property Aliases*.

Using Sample Data for WSDL Messages

Select the source location for the sample you want to add. Select a workspace resource, such as sample data, WSDL, or XSL file.

WSDL files define operations that receive an input message and, optionally, return an output message and a fault message. In the Participants or Interfaces view, you can add sample data values for message parts. When you are ready to test your process, you can load the sample message part values into your variables to simulate the actual messages exchanged by Web services. As you execute and debug your process, you can inspect the input, output, and fault data values as they are processed.

By adding sample data values to WSDL messages, you have a default set of data across all projects. The Project Explorer keeps track of all sample files and their locations and presents them to you when you are loading values into message variables.

Adding or Editing a Sample Data Value for a Simple Type Message Part

You can manage WSDL, schema, and XML data files in the Interfaces view to make your design work faster and easier. Use this dialog to add sample data to a WSDL simple-type message or edit sample data for the simple type variable.

In the Participants or Interfaces view, you can add a sample data value for a simple type message part. When you are ready to test your process, the sample message part value is automatically loaded into your variable.

You can also modify the value in Process Variables view as well as override the value during process simulation.

To add a single sample data value:

1. In the Participants or Interfaces view, expand a message with a part that is a simple type, such as `xsd:int`.

2. Double-click the variable to open the **Add Sample Data** dialog, as shown.

3. Type in a value and click **OK**. Notice that the value is displayed as a child node beneath the message part.

4. As needed, double-click the sample to edit it.

You can add multiple data values and then select one as the default by right-clicking on the sample data value. You can remove any values you do not want.

Generating a Sample Data File

Select the source location for the sample you want to add. This is a workspace resource such a sample data, WSDL, or an XSL file.

When you want to simulate process execution for a process with complex messages, you can generate sample data files for the messages. A sample data file is an XML file structured with message part elements or types that were defined in the schema declared in the WSDL namespace. The following is an example of a partial sample data file for a purchase order output message. Code similar to this example is automatically generated for you, based on the schema type or element.

```
<purchaseOrders
        xmlns="http://samples.cxdn.com/po"
        xmlns:tns="http://samples.cxdn.com/po">
  <purchaseOrder tns:orderDate="2005-09-05">
     <shipTo tns:country="US">
        <name>Jay Lor Jr.</name>
        <street>100 My Street</street>
        <city>My City</city>
        <state>CT</state>
        <zip>06484</zip>
     </shipTo>
     ...
  </purchaseOrder>
</purchaseOrders>
```

To generate sample data:

1. In Participants or Interfaces view, expand a message with a complex part, and select a message part for which you want to generate a sample XML data file.

2. Right-click on the message part and select **Generate Sample**.

3. In the XML Data Wizard, fill in your preferences for generating the sample data. For details, see *Using the XML Data Wizard*. Click **Next**.

4. Name the file, and save it to the `sample-data` folder in your orchestration project.

You can generate multiple sample data files for the same message part. You can name the files uniquely, or you can use the same filename and store the files in different folders. Process Developer keeps track of file locations.

Adding a Sample Data File to a WSDL Message

Select a workspace resource, such as sample data, WSDL, or XSL file.

You can add sample data files to your orchestration project and then add sample data files to WSDL messages that are complex types. You can select a default sample for all processes you create in Process Developer. On a process by process basis, you can add sample data to process variables in addition to or instead of using the samples added to a project or Interfaces view.

To add a sample data file to a message variable:

1. In Participants or Interfaces view, expand a message with a complex part, and select a message part for which you want to add a sample XML data file.

2. Double-click the complex part to open the **Project** dialog.

3. Select the source project location for the file.

4. Browse to the file, select it, and click **OK**.

The first file you add is set as the default sample, but you can add additional files and select one of them as the default.

Selecting a Default Sample Data File

You can add several sample data files for each message part in Participants or Interfaces view. You can select one sample file as the default file loaded into process variables.

In Participants or Interfaces view, expand a WSDL message. To select a default sample data file, right-mouse click on a sample file and select **Default Sample**.

Viewing the XML Structure of a Sample Data File

To view the XML text of a sample data file that added to a WSDL message in Interfaces view, right-mouse click on the data file name and select **Open**.

Removing a Sample Data File

To remove a sample data file from a WSDL message in Participants or Interfaces view, right-mouse click on the data file and select **Remove Sample**.

If you remove the file set as the default file, the first file in the list becomes the default.

Finding Where a WSDL Component is Used

In Process Developer, you can search for a WSDL component with the Find Where Used option.

The Find Where Used search helps you quickly locate a WSDL component reference in a Process Developer project file.

When you build BPEL processes in Process Developer, you reference the components of the WSDL files. You can search for a particular WSDL component to find the BPEL files that are referencing the component. This search is useful if a WSDL file changes because you can search for and update the references.

To Use Find Where Used:

1. Select a WSDL component, right-mouse click, and select **Find Where Used**.

2. Select the scope of the search:

   - **Workspace**: all project files in the workspace folder

   - **Working Set**: a named, customized group of files and folders

3. A list of project files matching the search criteria appears in the Search view. If this view is not visible (it's usually in the group of tabs in the bottom-right corner of the Process Developer window), select it from the **Window > Show View** menu.

4. Double-click a file to open it.

In the Search view, you can perform more search functions, such as search again, use a previous search, and search within search results. For more information, press F1 in the Search view.

Using Process Search

Process search helps you locate a WSDL component referenced in an ActiveVOS project file.

In Process Developer, you can search for the use of WSDL components within your BPEL files.

You can perform two kinds of searches:

- Search for a string or pattern using the Process Search tab in the **Search** dialog.

- Quick search for a WSDL component with the *Find Where Used* option. For more information, see *Finding Where a WSDL Component is Used*.

Process search helps you locate a WSDL component referenced in a BPEL file. If you do not remember the full name of the component, you can search using wildcards.

To perform a Process Search in the entire Workspace or in a Working set:

1. Select Process Search from the **Search** menu.
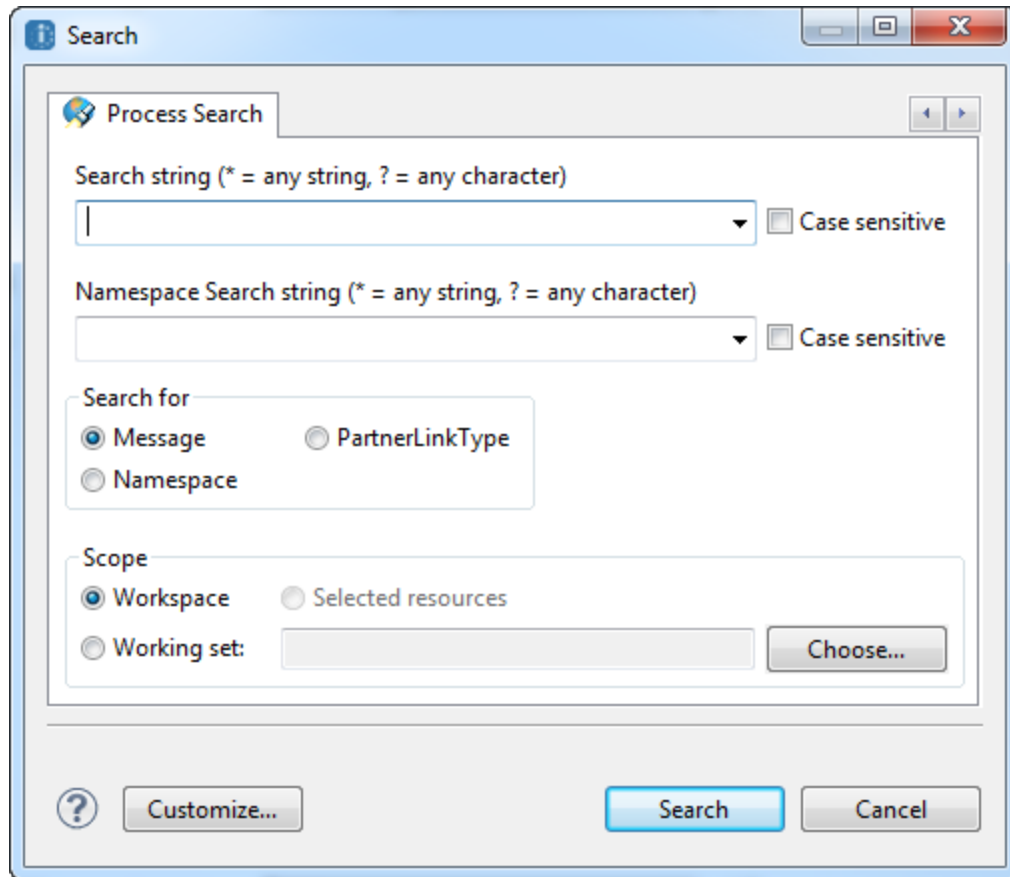
2. In the Search for panel, select the type of WSDL component:

    *   Message
    *   Partner Link Type
    *   Namespace

3. For a message or partner link type search, do the following:

    *   If desired, type in a search string, optionally with wildcards.
    *   If desired, further refine the search by providing namespace information.

4. For a namespace search, if desired, type in a search string, optionally with wildcards.

5. Select the scope of the search:

    *   Workspace, namely all BPEL files in the Project Explorer that were not imported from outside the workspace.
    *   Working set, which is a named, customized group of project files and folders.
    *   For details on using Selected Resources, `refer to the next procedure.`

6. Select **Search**.

7. A list of project files appears in the Search results view. If the Search view is not visible, select it from the **Window > Show View** menu.
    In the following example, the search criteria had "messages starting with the string "a" in all namespaces in the entire workspace," as indicated in the Search title bar.

Search ("Message *:a*" - 2 Occurrences in Workspace)
loanapproval_tutorial_complete.bpel - Active._Samples/Completed Tutorial
tutorial.bpel - Active._Tutorial
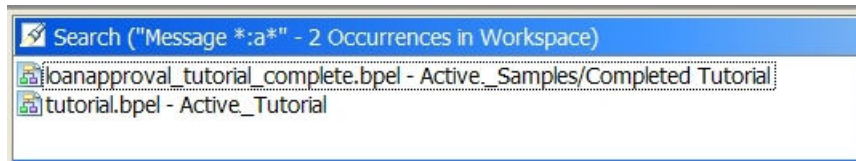
8. Double-click a file to open it.

To perform a Process Search with Selected Resources:

1. Display the Project Explorer view.

2. Select one or more files. When you open the **Search** dialog, the Select Resources option is selected.

3. Follow the procedure above, starting with Step 1.

In the following example, two files were selected in the Project Explorer before the **Search** dialog was opened. The search criteria was for all messages in all namespaces.

Search ("Message *:*" - 20 Occurrences in Selected resources)
store.bpel - wsr/bpel/bpr-ae-test/bpel (10 matches)
store-middleman.bpel - wsr/bpel/bpr-ae-test/bpel (10 matches)
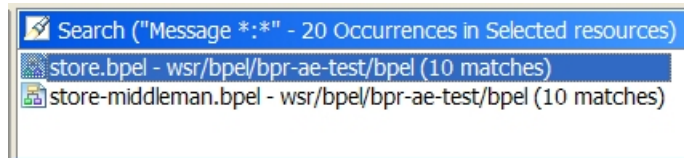
CHAPTER 5

Planning Your BPEL Process

To create a BPEL process, you need:

- An idea for an automated data exchange.
- A business partner with data to exchange with you.
- A flowchart type of plan that describes how you want to receive data, map data in assignments, invoke operations, and reply to your business partner.
- WSDL files describing messages, operations, and properties that are referenced by the process.
- WSDL definitions for each partner in the process that are in existence or that you'll create for your process.

Using Top-Down or Bottom-Up Process Design

You can use Process Developer to design a process using top-down, bottom-up, or a combination of techniques.

In the top-down technique, you sketch out your process by dropping down activities onto a canvas and creating links between them. You then add the information that binds the activities to an implementation and add decision-making to make your sketch a working process.

In the bottom-up technique, you use the definitions of the implementation that are available when you begin creating a process. The bottom-up technique requires complete and valid Web Services Description Language (WSDL) files. See *Using WSDL References for Efficient Design* for details.

In the combination technique, you can start a process with WSDL files already imported into your project. If you need additional WSDL files, you add them using the Interface Wizard. See *Creating a New Interface* for details.

Using WSDL References for Efficient Design

A BPEL process uses the definitions from Web Services Description Language (WSDL) files. WSDL files contain the namespace, partner link types, operations, and messages needed to define process activities, and WSDL files are required in order to create a valid, executable BPEL definition.

You can generate WSDL files in Process Developer or you can add them to the Project Explorer view of Process Developer before you begin creating a process definition.

- WSDL files provide several key productivity features:
- You can drop a WSDL operation onto the Process Editor to automatically create an Web Service interaction activity (Receive, Receive/Reply, Invoke, OnMessage, or OnEvent).
- You can automatically create the WSDL extensions required for a BPEL process if they do not already exist (partner link types, properties, property aliases).
- You can use the WSDL definitions across processes.

Creating WSDL Extensions for BPEL

A BPEL process depends upon WSDL extensions that are unique to BPEL. Process Developer can create these extensions automatically if they do not exist in your WSDL files. The extensions are partner link types, properties, and property aliases.

- A partner link type defines the roles played by a service and the port type provided by the service. For details, see *Partner Link Type*.
- Properties and property aliases are significant for correlating a group of messages in a process and can also be used for any process variable. For more information, see *Adding Variable Properties and Property Aliases*.

Starting a Process by Dropping an Operation onto the Process Editor

You can begin designing a BPEL process by using the key productivity features in Process Developer. Using either of the following starting points, you expose WSDL operations that can be dragged to the Process Editor canvas to automatically create activities:

- Add the Web services and partners used by the process, with BPEL extensions automatically generated. See *Participants*.
- Use your existing WSDL to create BPEL extensions. See *Interfaces, Service References, and Local WSDL*.

Importing WSDL Schema and Other Resources

Define the location of the resource file containing the target namespace.

A BPEL process must contain namespaces that point to WSDL and schema locations. Because one namespace can have many WSDL or schema files associated with it, Process Developer identifies the correct file by importing it.

You can import other resources you need for BPEL processes, such as XSL style sheets for use in creating expressions and XML files used in resource catalog custom functions.

You can add imports manually or automatically.

Automatically Importing WSDL and Schema Locations

Your WSDL and schema files can be located in many places and can share the same namespace. When you specify a WSDL or schema location, Process Developer adds the namespace declaration to the BPEL process and ensures that the namespace points to the correct reference.

Process Developer takes care of WSDL location and namespace declarations for you when you add a WSDL file to Project Explorer and then create a Receive or Invoke activity by starting with an existing operation or a new operation. For basic information on using the convenience of interfaces, see *Participants* and *Interfaces, Service References, and Local WSDL*.

See also *Creating an Activity by Starting with a WSDL Interface*.

Manually Importing WSDL Schema and Other Resources

Define the location of the resource file containing the target namespace. Select a workspace resource, such as sample data, WSDL, or XSL file.

If you import and open a BPEL process into Process Developer before adding a WSDL file to Project Explorer, Process Developer reports unresolved references to port types because the WSDL is unknown.

The recommended practice for adding WSDL is to use Participants. For details, see *Using the Participants View*.

You can add WSDL location and namespace manually. You can also add schema and other resources as follows.

1. From Outline view, select **Imports**.

2. Right-mouse click and select **Add > Declaration > Import**.

3. From the Import Types list, select a type.

4. If you select XSL, the Type URI of the file is added. This type identifies the encoding language used in the file as shown:
    `http://www.w3.org/1999/XSL/Transform`

5. In the Location group, do one of the following:

    *   Select **Project** and browse to the Project Explorer folder containing the file.

    *   Select URL and type in a Web location of a file, such as:
        `http://www.tempuri.org/myFile.wsdl`

        or

        `http://www.tempuri.com/service.asmx?WSDL`

        Note that when the process is deployed and running, this URL must be available.

    Notice that the target namespace is displayed for the WSDL or schema file. If you paste a pathname or URL into the WSDL (or Schema) file location field, you can press the Tab key to see the target namespace. Click **OK** when you are through.

6. In the Outline view, right-mouse click the Imports node, and select **Refresh Imports**.

To modify an import, double-click it, or in Properties view, click the `Dialog (...) Button` next to Location.

Process Developer adds the target namespace to the BPEL process and associates the namespace with a default prefix. Also, the port types, operations, and partner link types defined in the WSDL are available for use in the process.

The following illustration shows how the Process Developer displays a WSDL location and namespace. Note that Process Developer adds a new namespace prefix with a default name `ns1`. See *Namespace Prefix and Declaration* for more information.
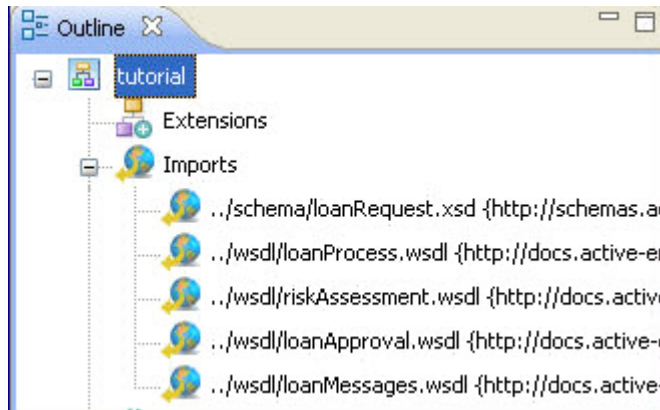
See also *Refreshing Imports*.

Refreshing Imports

The Imports node in Outline view displays the location of all WSDL, schema, and other resources referenced in your process. If you modify an import or create a new resource, you can refresh Imports to update the reference.

Refreshing Interfaces does not refresh the imports information.

Tip: If you import and open a BPEL process in Process Developer before adding a WSDL file to Project Explorer, Process Developer reports unresolved references to port types, operations, and variables because the WSDL is unknown. You can fix these errors as follows:

1. Add the WSDL files to Project Explorer, as described in *Importing a Local WSDL*, *Creating a New Interface*, or *Importing a Service Reference*.

2. In the Outline view, right-mouse click on Imports, and select **Refresh Imports**.

Deleting an Import

Delete imports from this BPEL process. The associated namespace prefix and URI are not deleted. You can manually delete the namespace, if desired.

You can delete an imported WSDL, schema or other resource locations by right-mouse clicking an import in Outline view and selecting **Delete**. From the menu bar, you can also select **Process > Delete > Import**.

The associated namespace prefix and URI are not deleted. You can manually delete the namespace, if desired.

Namespace Prefix and Declaration

A BPEL process must contain namespaces that point to WSDL locations. To add WSDL locations, you use the Imports function, as described in *Importing WSDL, Schema, and Other Resources*.

After adding an import, you can rename the default prefix for the namespace, as follows.

1. In Outline View, select a *Namespace*.

2. In the Properties view, type in a prefix to use as a shortcut definition for the namespace. The prefix is mapped to the URI reference displayed.

If you delete an import, you must manually delete the namespace associated with it.

Declaring Extensions

An extension declaration allows you to add a new construct to a BPEL process. Process Developer ignores it, since it can not understand it.

You can extend the definition of the BPEL language by declaring an extension. An extension declaration allows you to add constructs such as a new activity type or new processing instructions that are outside of the WS-BPEL 2.0 specification.

Process Developer and the Process Server use extensions. The Process Developer extensions are:

- Query handling for automatically creating XPath for Copy To targets and for disabling the bpel:selectionFailure fault. (See *Using the Process Developer Create XPath Extension* and *Using the Process Developer Disable Selection Failure Fault Extension*.)

- Suspend, break, and continue activities. (See *Overview of BPEL Activities*.)

- Process level compensation/termination .(See *Creating a BPEL Process as a Service for Another BPEL Process*.)

- Implicit scope variables.

- Loopback links. (See *Using Links*.)

When you use one of these extensions, the extension namespace is automatically added to the Outline view. However, you can add the extension namespace manually before you use the extension, if desired.

To add your own extension definition:

1. From the Outline view, select Extensions, and right-click to select **Add > Declaration > Extension**.

2. In the **Extension Definition** dialog, select User Defined Extension Namespace.

3. Type in a namespace referencing your BPEL extension.

4. Check **Must understand** if required. Note that when this is selected and you use the extension in your process, Process Developer reports an error when you save the process. The process cannot be simulated or deployed. If you do not select Must understand, your extensions are ignored by the Process Server.
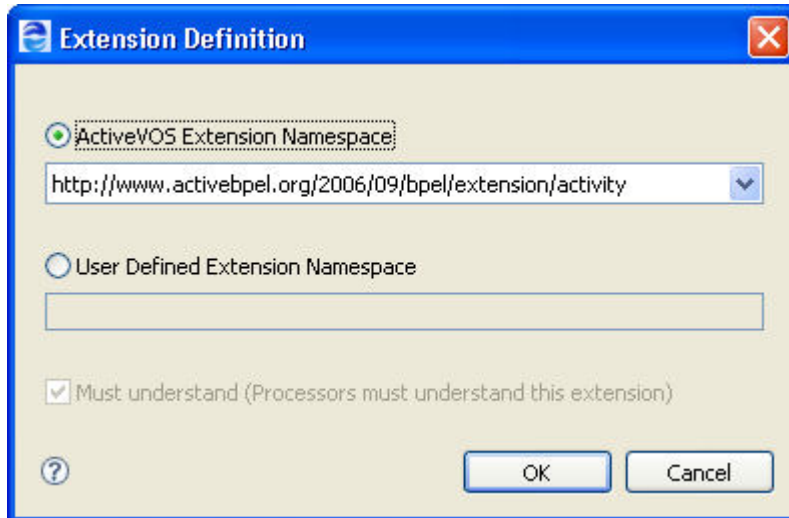
Using the Process Developer Create XPath Extension

The Process Developer `Create` XPath extension allows an executing process to automatically create a location path for a non-existent node in a complex variable. You can use this extension on a process-by-process basis or set a preference to include it in all processes. When you use the extension, an extension namespace and process attribute are added to the process as follows:

```
<bpel:process ...
    xmlns:ext=
        "http://www.activebpel.org/2006/09/bpel/extension/query_handling" ...
    ext:createTargetXPath="yes" ...>
  <bpel:extensions>
   <bpel:extension mustUnderstand="yes" namespace=
      "http://www.activebpel.org/2006/09/bpel/
       extension/query_handling"/>
  </bpel:extensions>
...
```

The process extension is enabled during simulation in Process Developer and enabled for execution on the Process Server.

To enable or disable this extension for an individual process:

1. In the process Properties view, select the All tab.

2. In the Create XPath row, select Yes or No from the pick arrow.

The extension is added to the Outline view, and its elements and attributes are added to the BPEL source code.

Using the Process Developer Disable Selection Failure Fault Extension

The Process Developer Disable Selection Failure extension allows an XPath query in the From clause of an assignment Copy operation to return an empty node set. If it does, the target node for the assignment is deleted. You can use this extension on a process-by-process basis or set a preference to include it in all processes. When you use the extension, an extension namespace and process attribute are added to the process as follows:

```
<bpel:process ...
    xmlns:ext=
        "http://www.activebpel.org/2006/09/bpel/extension/query_handling" ...
    ext:disableSelectionFailure="yes" ...>
  <bpel:extensions>
   <bpel:extension mustUnderstand="yes" namespace=
      "http://www.activebpel.org/2006/09/bpel/
       extension/query_handling"/>
  </bpel:extensions>
...
```

The process extension is enabled during simulation in Process Developer and enabled for execution on the Process Server.

To enable or disable this extension for an individual process:

1. In the process' Properties view, select the All tab.

2. In the Disable Selection Failure row, select Yes or No from the pick arrow.

The extension is added to the Outline view, and its elements and attributes are added to the BPEL source code.

Declaring Extension Elements and Attributes

An extension element declaration allows you to add a new attribute to a BPEL construct. Process Developer ignores it, since it can not understand it.

The WS-BPEL 2.0 specification allows for extensions to the basic BPEL language in several ways, including extension elements and extension attributes for BPEL constructs.

For example, if you open a process containing elements and attributes that are not defined in the specification, its extension details are preserved by Process Developer.

If the extension details include a Must Understand attribute set to Yes, you cannot simulate or remote debug your process since Process Developer does not understand how to execute the process. In this case, Process Developer declares an unknown extension as an error.

If the Must Understand attribute is set to No, the extensions are ignored when you simulate and debug your process.

You can add extension elements and attributes to any WS-BPEL 2.0 process; however, if you enable the Must Understand attribute, you cannot simulate or deploy the process to A Process Server.

To add or modify an extension attribute:

1. From the Outline view, select the BPEL construct that uses an extension attribute. Note that almost every construct, including the process element, partner links, activities, correlation sets, fault and event handlers, contain the Extension Attributes property in the Properties view.

2. In the Properties view, select the **Show Advanced Properties** button in the toolbar to display all properties.

3. Select the `Dialog (...) Button` at the end of the Extension Attributes row.

4. If your imported process contains extension attributes, they are displayed in the **Extension Attributes** dialog. Do one of the following:

   - Select an attribute from the list and select **Edit**.

   - Select **Add** to create a new attribute.

5. In the **Extension Attribute** dialog, do the following:

   - Type in a namespace or select from a list of existing extension namespaces.

   - Add or edit the attribute Name.

   - Add or edit the attribute Value.

   - Select the Declare this extension check box if the extension namespace is new.

   - Check **Must understand** if a BPEL engine must understand the attribute value in order to correctly execute the process. Note that the Process Developer engine cannot understand the attribute definition or value, and you cannot simulate or deploy the process to a Process Server.

   - Check **Must understand** if a BPEL engine must understand the attribute value in order to correctly execute the process. Note that the Process Developer engine cannot understand the attribute definition or value, and you cannot simulate or deploy the process to a Process Server.

   The following illustration shows an example.
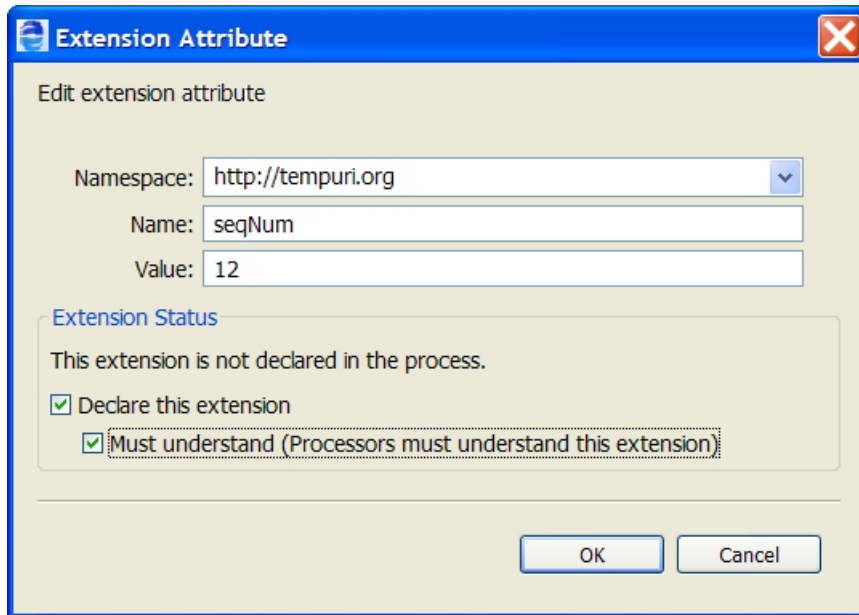
6. Click **OK**, and notice that the attribute is listed in the **Extension Attributes** dialog where you can edit and remove attributes from the list.

Tips:

- After you add an extension attribute, Process Developer adds a prefix for the namespace. If desired, select Namespaces from the Outline view and rename the prefix to something more meaningful.

- The extension attributes are displayed in the Properties view of the BPEL construct, in the order they were created.

- To reset the Must Understand attribute, select the Extensions namespace declaration from the Outline view and change the value in the Properties view

To add or modify an extension element:

1. From the Process Editor or Outline view, select the BPEL construct that uses an extension element. Note that almost every construct, including the process element, partner links, activities, correlation sets, fault and event handlers, contain the Extension Elements property in the Properties view.

2. In the Properties view, select the **Show Advanced Properties** button in the toolbar to display all properties.

3. Select **Dialog (…)** at the end of the *Extension Elements* row.

4. If your imported process contains extension elements, they are displayed in the **Extension Elements** dialog. Do one of the following:

    - Select an element from the list and select **Edit**.

    - Select **Add** to create a new element.

5. In the **Extension Element** dialog, do the following:

    - Type in fully qualified valid XML syntax in the Element text box. Process Developer validates the XML before enabling the **OK** button.

    - Select the Declare this extension check box if the extension namespace is new.

- Check **Must understand** if a BPEL engine must understand the element value in order to execute the process. Note that the Process Developer engine cannot understand the element definition or value, and you cannot simulate or deploy the process to A Process Server.
  The following illustration shows an example:
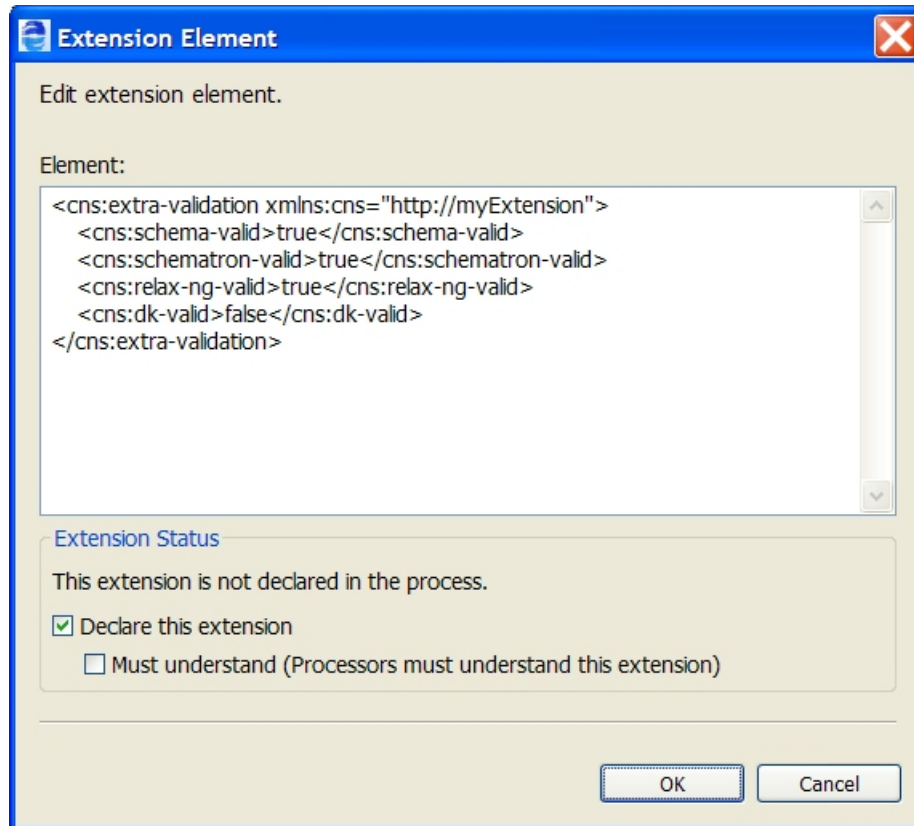
6. Click **OK**, and notice that the element is listed in the **Extension Elements** dialog where you can reorganize the list and remove elements from the list.

Tips:

- Reorganize the extension element entries to reflect the order in which they should be executed

- To reset the Must Understand attribute, select the Extensions namespace declaration from the Outline view and change the value in the Properties view

Understanding BPEL Process Structure and Properties

A BPEL process has the following structural parts:

- Process Element and Properties
- Variables
- Partner Links
- Activities

- Fault Handlers

- Compensation Handlers

- Message Exchange Declaration

- Importing WSDL, Schema, and Other Resources

- Declaring Extensions

- Correlation

A BPEL process is represented graphically in the Process Editor, as described in *Using the Process Developer Process Editor*. In addition, you can view BPEL source code, as described in *BPEL XML Source and Implicitly Added Activities*.

Process Element and Properties

After you create a new BPEL file in Process Developer, add an activity to the Process Editor canvas, and save your file, Process Developer creates the underlying XML source file. The XML source begins with the `<process>` element.

The XML Syntax for an executable process element looks like this:

```
<process name="NCName" targetNamespace="anyURI"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  exitOnStandardFault="yes|no"?
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
```

If you use Process Developer extensions, other namespaces and process attributes can be added, as described in *Declaring Extensions*.

Click on a blank part of the Process Editor canvas to put the process as a whole in focus. In the Properties view, set the Process properties as desired.

Property	Description	Default Value
Generate Prefixed Source	Specifies that BPEL XML source code elements are generated with the `<bpel:...>` prefix. Enabling this property ensures that the attribute `xmlns:bpel` will not be null.	yes
Write Port Type	Add port type attribute to activity definition for receives, replies, onEvents, onMessages, and invokes.	no
Abstract Process	Specifies whether the process is abstract or executable. For more information, see *Creating an Executable vs. an Abstract Process*. This setting can be specified as a preference for all new processes. See *Process Developer Preferences*.	no
Abstract Process Profile	List of default profiles referenced in the WS-BPEL 2.0 specification	`http://docs.oasis-open.org/ wsbpel/2.0/process/ abstract/simple-template/ 2006/08`
BPEL Namespace	Specifies the BPEL language namespace for an executable or abstract process. The default is for executable processes.	`http://docs.oasis-open.org/ wsbpel/2.0/process/ executable`

Property	Description	Default Value
Comment	Optional property for adding an HTML-tagged annotation to the `<process>` element. You can also add a comment to any activity, link, or container.	none
Create XPath	This property is a Process Developer extension to WS-BPEL 2.0, described in *Declaring Extensions*.	yes
Disable Selection Failure	This property is a Process Developer extension to WS-BPEL 2.0, described in *Declaring Extensions*.	yes
Documentation	Optional property for adding annotation to the `<process>` element. You can also add documentation to any activity, link, or container. See *Adding Documentation to a Process*.	none
Exit on Standard Fault	If this property is set to *yes*, then the process exits immediately as if an exit activity has been reached, when a WS-BPEL standard fault other than `bpel:joinFailure` is encountered. If this property is set to no, the process can handle a standard fault using a fault handler. Processes can be suspended on an uncaught fault. This behavior takes precedence over the Exit on Standard Fault setting.	no
Expression Language	Specifies the language used in building variable and other expressions. The default is XPath 1.0. There is also built in support for XQuery 1.0 and Javas-Script 1.5 Other languages can also be added.	`urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0`
Extensions	Indicates element and attribute extensions to the WS-BPEL 2.0 specification that are not understood by Process Developer exist in the process. Extensions can also exist in most other BPEL constructs, such as partner links and activities. For details, see *Declaring Extension Elements and Attributes*.	none
Links are Transitions	By default, allows links to completed activities. For details, see *Process Developer Extension for Links*.	yes
Message Exchanges	A process or scope property that can be selected as an attribute of a receive or onMessage and its matching replies	(none)
Process Level Compensation/ Termination	This is a Process Developer extension to WS-BPEL 2.0. Specifies whether the process as a whole can be compensated and terminated by platform-specific means. The process instance can be compensated after normal completion. For details, see *Creating a BPEL Process as a Service for Another BPEL Process*. If you set this to `yes`, the Process Editor displays new tabs. For details, see *Process Editor Compensation and Termination Handler Tabs*.	no

Property	Description	Default Value
Process Name	You can specify a process name that differs from the BPEL file name. This is useful if you create different or updated versions of the same process. You can name the BPEL file with meaningful version information, but can use the same internal process name. The process name property, not the BPEL filename, is known to the server engine.	name of the BPEL file
Query Language	Specifies the XML query language. The default is XPath 1.0. Also available for future standards as they evolve.	urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0
Suppress Join Failure	A join condition is a property of all activity types and provides status about an incoming link. A join failure indicates that link execution failed. At the process level, this attribute specifies whether a join failure is suppressed for the whole process. Can be overridden at the activity level and can be specified as a preference for all new processes. See *Process Developer Preferences*.	yes
Target Namespace	The XML namespace that refers to the required WSDL information. This is a required property.	none
Image Generation	See *Generate Deployment Image*	Automatic

Partner Links

A BPEL process describes a flow of interactions between the process and services. Each interaction describes what role the process and services play at that step in the flow and what data can be manipulated by the parties in those roles.

The constructs used to define roles are:

- Partner link type
- Partner link

You can define and add these roles in Process Developer. For more information, see *Participants*.

Variables

The BPEL process receives, manipulates, and sends data through XML variables. For example, the process receives a purchase order message from a buyer and puts the message in an input XML variable, and from there it can be copied to another operation.

Variables are defined in one of the following ways:

- WSDL message types
- XML schema types
- XML schema elements

Variables allow processes to maintain state data, (such as time outs) and process history based on messages exchanged.

Activities

Activities are the processing steps, performed in the order described by the flow diagram you design in Process Developer.

In Process Developer, there are several types of activities:

- Basic activities to exchange and assign data and define execution steps
- Containers to structure activities and handle faults and compensation
- Event Handlers

For more information, see *Implementing a BPMN Task or Event in BPEL*.

Fault Handlers

A fault handler defines the activity that the process must perform in response to an error condition. For more information, see *Fault Handling*.

A fault handler defines the activity that the process must perform in response to an error condition.

Compensation Handlers

Compensation is the process of reversing or providing an alternative for a successfully completed activity, especially when a fault occurs. For more information, see *Compensation*.

BPEL XML Source and Implicitly Added Activities

Process Developer generates BPEL XML source code for each construct you add to the Process Editor. You do not need to save your file to see the source. Select the Source tab of the Process Editor to view or edit the source code.

You may notice that Process Developer automatically wraps process-level activities in a flow container if you do not and if a sequence or other container is not present. If Process Developer adds a top-level container for you, it does not appear on the Process Editor canvas or in the Outline view.

Understanding BPEL Process Lifecycle

A BPEL process instance has a beginning, middle, and end, which are all defined by activities. The process must begin with one of the following:

- Receive activity
- Pick activity (with an `onMessage` event)
- Flow or Sequence activity containing one or more Receive or Pick activities

In a Receive or Pick, the *Create Instance* property must be set to `Yes`. The middle of the process consists of activities, running in sequence or concurrently.

The process instance ends in one of the following ways:

- An activity defines that the process is complete
- A fault reaches a process scope, and the process exits

Creating an Executable vs. an Abstract Process

In Process Developer, you can build two kinds of business processes:

- An *executable process* contains all the actual message data, operations, and partner information required for a running process. It uses the full power of data assignment and selection. Process Developer creates executable processes by default. You can simulate, deploy, remote debug, and execute an executable process.

- An *executable process* contains all the actual message data, operations, and partner information required for a running process. It uses the full power of data assignment and selection. Process Developer creates executable processes by default. You can simulate, deploy, and execute an executable process.

- An *abstract process* is a business protocol description document that tells your business partners what a process will look like when it runs. It outlines the steps of a process without actually filling in all the actions and data required for a running process.

An abstract process lets you share information with your partners without exposing how you intend to use data from your service. It gives your partners a way to understand how to create their WSDL files to work in your process and to tell them what information you plan to send them when the process runs. An abstract process outlines the type of data to be exchanged, as well as activities, time limits, error handling, and other meaningful pieces of information for the process.

You cannot simulate, deploy, remote debug, or execute an abstract process.

You cannot simulate, deploy, or execute an abstract process.

See also *Creating an Abstract Process* and *Tips for Working with Abstract Processes*.

Creating an Abstract Process

To create a process that is not immediately intended for simulation, remote debugging, or deployment, you can define the process as abstract.

To create a process that is not immediately intended for simulation or deployment, you can define the process as abstract..

To create an abstract process:

1. Select **File > New > BPEL Process**.
2. Select your project folder, and in the File name field, type in a name for your BPEL file. The `.bpel` extension is automatically added.
3. Select **Advanced** to view properties that you can set for this process.
4. Select Create as Abstract Process.
5. Click **Finish**.

You can make an executable process into an abstract one, and vice versa, by changing the process's Abstract Process property from No to Yes.

See also *Tips for Working with Abstract Processes*.

Tips for Working with Abstract Processes

As described in *Creating an Executable vs. an Abstract Process*, you can create a process not immediately intended for execution. When working with abstract processes, consider the following tips:

- For an abstract process, the namespace is: `http://docs.oasis-open.org/wsbpel/2.0/process/abstract`

- You can set an Abstract Process Profile in the process Properties view. The default profiles listed are referenced in the WS-BPEL 2.0 specification.

- You can use an activity unique to abstract processes. For details, see *Opaque*.

- You can use an *opaque from-spec* assignment in a copy operation. For details, see *Assign*.

- You can make an abstract process into an executable one by changing the process's Abstract Process property from Yes to No. If your process contains an opaque activity, you are warned to replace it with a different activity.

- If desired, you can follow the guidance in the WS-BPEL 2.0 specification for hiding syntactic elements in your process. See the examples below.

WS-BPEL XML Fragment Examples

Example 1 - Variable Declaration

```
<variable name="commonRequestVar" element="##opaque" />
```

Example 2 - Use of Invoke Activity

```
<invoke partnerLink="homeInfoVerifier"
    operation="##opaque" inputVariable="##opaque"
    ext:uniqueUserFriendlyName="request verification" />
```

Example 3 - Use of Opaque Activity

```
<opaqueActivity template:createInstance="yes">
    <documentation>...</documentation>
</opaqueActivity>
```

Creating a BPEL Process as a Service for Another BPEL Process

A BPEL process uses an invoke activity to call out to a Web service to perform a task. You can create a BPEL process for this purpose.

You can choose between two different invocations of a process:

- Standard **process** that behaves like a normal endpoint reference
When you create a process deployment descriptor, you can select any process as a static endpoint reference for a partner role. This avoids the overhead of the server composing a SOAP message for dispatch through the standard invocation layer. The message is sent directly to the engine instead.

- **Subprocess** of the calling process that is eligible for process-level compensation and termination handling
When you create a process deployment descriptor, you can select any process:subprocess as a static endpoint reference for a partner role. This lets the subprocess participate in process-level compensation and termination handling.

More about Subprocesses

A subprocess is subject to the fault and compensation handling rules of the enclosing scope of the parent process from which it is invoked. A subprocess is compensated whenever either the fault handler or the compensation handler for the scope enclosing the invoke activity is run. This work is performed implicitly. However, you can enable process-level compensation and termination for the subprocess to select what compensation and termination tasks to do. For a discussion of compensation, see *Compensation*.

The enclosing scope for the invoke activity does not complete until the subprocess completes. If the subprocess is compensatable when it completes, the enclosing scope compensates the subprocess during its compensation.

If the subprocess throws a fault, the invoke activity of the parent process faults.

Creating a Process as a Service

To create a BPEL process as a service, you must do the following:

1. Add a WSDL file to Project Explorer for the BPEL process. There are many ways to create a WSDL file. *See Interfaces, Service References, and Local WSDL*.

2. Create the BPEL process to be used as a service.

3. If desired, set the process property Process Level Compensation/Termination to Yes. Two additional tabs are added to the Process Editor, and you can add compensation and termination activities to them. The following illustration shows the tabs that are added to Process Developer.
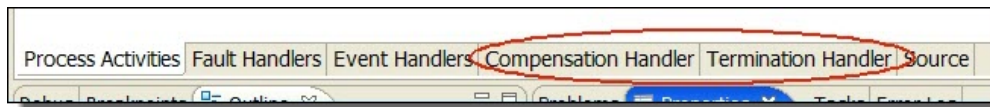
4. In the Process Deployment Wizard, identify a service name for the My Role partner link, just as you would for any process.

5. Create a business process archive (.BPR), as described in *Deploying Your Processes*.

6. Deploy the process to the server.

7. Create the main process that invokes the BPEL process using an invoke activity.

8. In the Process Deployment Wizard for the main process, do the following for the Partner Role partner link:

   a. Select process or sub-process as the invoke handler.

   b. Select static as the endpoint reference.

   c. Select the **Dialog** button to fill in the sub-process information.

9. Deploy the main process.

When the main BPEL process is running, and it invokes another BPEL process, the invoked process has the status of a subprocess, with its own state information. The main process awaits the completion of the subprocess before continuing its own execution. For more information on actively running processes, see the *Process Console Help*.

Message Exchange Declaration

Use the Message Exchange Declaration dialog to add, edit, or remove message exchange declarations for the process as a whole or for the current scope.

A message exchange value binds a synchronous receive with its reply. A message exchange value makes an explicit match between a receive and a reply, eliminating any ambiguity among concurrent execution of synchronous receives with the same partner link, port type and operation.

A message exchange is a process or scope property that you can select as an attribute of a receive, onMessage, or onEvent and their matching replies. Declare the message exchange value in this dialog. Then select the value, as needed, in the appropriate receive/reply pairs, onEvent/reply pairs, or onMessage/reply pairs.

When to Use This Declaration

The process and the child scope for a parallel forEach activity implicitly declare a default message exchange value. If a receive or a reply activity does not have a message exchange value explicitly declared on it, it has the default value that is provided by either the process or the parallel forEach's scope, depending on where the receive or reply is nested. You can still use explicit message exchange values on receives, replies, and onMessages within a parallel forEach.

The message exchange value can also be useful in multiple receive/reply, onEvent/reply or onMessage/reply pairs in a flow that have the same partner link, port type, and operation. By adding a message exchange value to the pairs, you can avoid receiving the bpel:conflictingRequest fault.

A process is required to reply to any synchronous message exchanges (receives or onMessage) it receives during execution. If there isn't one, the process faults has a bpel:missingReply fault. Each scope that declares a message exchange value tracks its active message exchanges to ensure that a reply executes prior to the scope completing.

When you add a new message exchange value, the default name is MessageExchange_1. You can edit this value, if desired.

If a message exchange declared in an inner scope has the same value as one declared an outer scope, or in the process, the inner scope's declaration overrides the outer scope's or process' declaration. Receives, onEvents, onMessages, and replies in different scopes can be matched only by using the outermost scope's message exchange declaration.

CHAPTER 6

Participants

This chapter includes the following topics:

What are Participants

Participants are the people and services that the process interacts with over the course of its execution. Communication to or from a participant is always through a Web service interface.

Participants simplify the BPEL details you need to provide for an interface. By creating participants, you can skip over some technical details in BPEL as Process Developer generates details for you.

Here are the participant types:

- **Process service consumers**
  Each process interacts with at least one participant: the participant that sends the service request message that starts the process. This participant is the first process service consumer. You can also create additional consumers if there are additional participants that use services provided by a process. Consumers can also provide callback service. These allow a process to request that the consumer perform operations. In BPEL, the process service consumer is equivalent to a partner link whose myRole property has been set.

- **Partner service providers**
  If the process needs to invoke a service to do some work for the process, it calls a service that is provided by a partner service provider. The process can also make callback services available to a partner service provider. The interface for such a callback is listed with the partner service provider that can use it. In BPEL, this participant is the equivalent to a partner link whose *partnerRole* property has been set.

- **Human task participants** (on-premises only)
  If the process needs people to accomplish some of its steps, it uses people activities. A human task participant represents the role of the person who will complete some of the activities of the process. Each human task participant lists the tasks that its role works on. In BPEL4People, a *Logical People Group* is created for each human task participant.

  For details, see Using the *Participants View*.

  For details on using the Participants view in conjunction with HTML form development, see *Adding a New Service Operation for a Form or a Task*.

Using the Participants View

The Participants view is the key starting point for developing a BPEL process. It guides you to provide the key ingredients for a process: the roles that the process and partner services play and the interface for each role. Further, once you define an interface, you can drag its operation to the Process Editor canvas to create process activities. For example, a receive activity is created from the interface operation associated with a process service consumer.

The following illustration shows an example of participants and their interfaces:
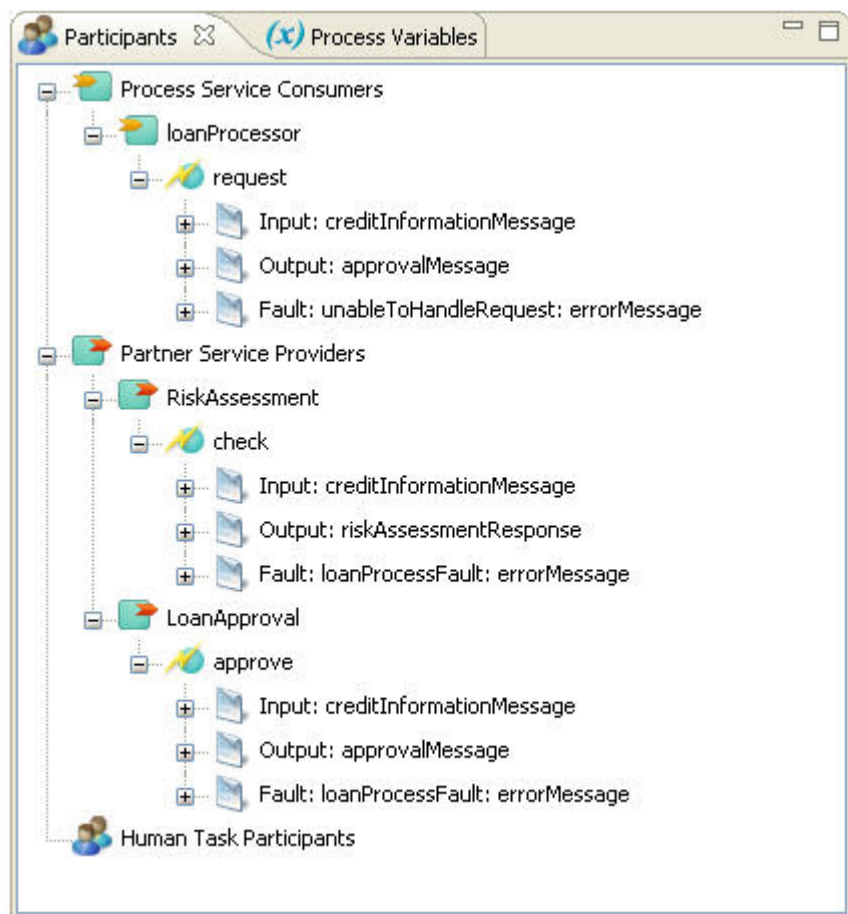
The Participants view is empty unless a process is open. The view is process-based; that is, it shows participants only for the process in focus.

In this view, you can do the following (on-premises only):

- Create new process and service interfaces. See *Creating a New Process Service Consumer Interface* and *Creating a New Partner Service Interface*

- Create new human task participants and their tasks for use in people activities.

- Drag an operation (or task) to the Process Editor to create appropriate activities. See *Creating New Activities from the Participants View*.

Creating a New Process Service Consumer Interface

A Process Service Consumer is a participant Web service that is used in building a receive and/or reply activity. It consists of a name and an interface.

You can create a service consumer in stages:

- Enter a name or accept the default name, Process_Consumer.

- If interface resources are available, you can define the interface. If they are not, you can add them later, in the Properties view.

- By completing the participant details, you are essentially creating all the required elements needed for a process' receive and reply activities.

Creating a New Partner Service Interface

A Partner Service Provider is a participant Web service that is used in building an invoke activity. It consists of a name and an interface.

You can create a partner service provider in stages:

1. Enter a name or accept the default name, Provider.

2. If interface resources are available, you can select the interface. If they are not, you can add them later, in the Properties view

When you complete the participant details, you are essentially creating all the elements needed for a process' invoke activity.

Creating a New Callback Interface

Select or create the port type and operation for a participant to call back.

Each Web service in a process, represented by receive, reply, and invoke activities, must have a service interface. The interface describes the operations provided by the service.

You can choose to add a callback interface, which allows the service provider to invoke operations on the service consumer. For example, in a service interface, the process sends an order and the partner replies with an order ID. In a callback interface, the partner can send an order shipment notice.

When you add a callback interface, Process Developer generates a standard BPEL partner link type for that participant and stores it in a WSDL. This makes the participant relationship clear for future users of the WSDL and allows new processes that use the service interface to automatically see the callback interface.

The details for creating a callback interface are the same as creating a service interface. See:

- *Creating a New Process Service Consumer Interface*

- *Creating a New Partner Service Interface*

Tips:

- Process Developer adds the BPEL-required details, namely partner link type definitions, to a new WSDL. It adds definitions that need to be seen by clients of the process in a file named [process].public.wsdl. It places definitions that are private to the process (that is, partner link types for partner services) in a file named [process].public.wsdl.

- The Callback participant displays as a child participant in the Participants view.

See also *Creating New Activities from the Participants View*

Clearing a Service Interface from a Participant

Remove a service interface from a participant.

You can clear a service interface from a participant from the Properties view.

- From the Participants view or the Outline, select a participant.

- In the Properties view. select the Dialog button in the Service Interface row.

- In the **Service Interface** dialog, select the **Clear** button in the Interface row.

Creating New Activities from the Participants View

After creating participants and their interfaces, you can drag an operation or task to the Process Editor to create activities. These activities contain the required participant name and operation. You can then use the activity's Properties view to provide data assignments.

The following table shows the activities that the Process Developer creates when you drop an operation onto the Process Editor canvas.

Drag an operation from this participant's interface	To create this activity or activity pair
Process Service Consumer one-way operation	Receive, onMessage (if Pick is selected), or onEvent (if event handler is selected)
Process Service Consumer two-way operation	Receive-Reply, onMessage-Reply, onEvent-Reply
Process Service Consumer Callback	Invoke
Partner Service Provider	Invoke
Partner Service Provider Callback	Same activities as process service consumer for one-way and two-way operations

See also:

- *Creating a New Process Service Consumer Interface*

- *Creating a New Partner Service Interface*

- *Creating a New Callback Interface*

- *Creating a new Human Task Participant* in the *Human Tasks* section of this help

Creating a New Variable From an Activities Property View

Create a new process variable for a receive or other activities.

Create a new process variable for a receive onMessage, onEvent, reply, or invoke activity as follows:

1. Select the Properties view of an activity.

2. Open the Data, Input or Output tab of the activity.

3. For the Assignment Type, select Single Variable.

4. In the Variable list, select New Variable.

The variable name suggested is one of the following:

- The qualified name of the schema element for a single-part element WSDL message

- The local name of the message if the single-part element is an XSD type, such as string

- For a multi-part WSDL message, the local name of the message

In the Scope field, select the process or a scope to make the variable global or local.

The Variable Type is derived from the WSDL message definition.

What are Partner Link Types and Partner Links

Select partner links to include in a partner definition. A partner link can be used in only one partner definition.

For an easy way to create and use partner link types and partner links, see *What are Participants?*

A BPEL process describes a flow of interactions between the process and services. Each interaction describes what role the process and services play at that step in the flow and what data can be manipulated by the parties in those roles.

BPEL defines constructs to identify roles and relationships used in interactions.

The constructs are partner link and partner link type. A partner link describes the roles that a process and service play as well as what data they can manipulate in that role. A partner link is defined by its partner link, as shown in the following diagram.

The partner definitions and usage in a BPEL process do not refer to the specifics of Web services. Since a BPEL process is a reusable definition that can be deployed in different ways, the definitions are abstract. The addressing, security, and other specifics of Web services are taken care of at deployment time. When a process is deployed, every partner role in a partner link for a BPEL process instance is assigned a unique endpoint reference.

For more details on each concept, see the following:

- *Partner Link Type*
- *Partner Link*

Partner Link Type

Create a standard BPEL activity, such as a receive, reply, or invoke. Add a partner link type definition to a new WSDL file or to an existing WSDL file.

A partner link type describes the kind of message exchange that two WSDL services intend to carry out. A partner link type characterizes this exchange by defining the roles played by each service and by specifying the port type provided by the service to receive messages appropriate to the exchange.

For an easy way to create and use partner link types and partner links, see *What are Participants?*.

A partner link type can include one role or two roles.

- If a partner link type contains two roles, each of the services must implement its role by providing the specified port type. Using two roles indicates that there is a requirement for the calling service to receive some type of callback from the target service in the course of the conversation. For example, a service is invoked with a one-way operation (input message only). When it is ready to reply, it must send back a message to the process' receive activity.

- If a partner link type contains only one role, no restriction exists on the calling web service regarding roles. The service described by the single role can complete the conversation simply by having its operation invoked without the need for issuing a callback.

The following illustration shows an example of a partner link type defining two roles.
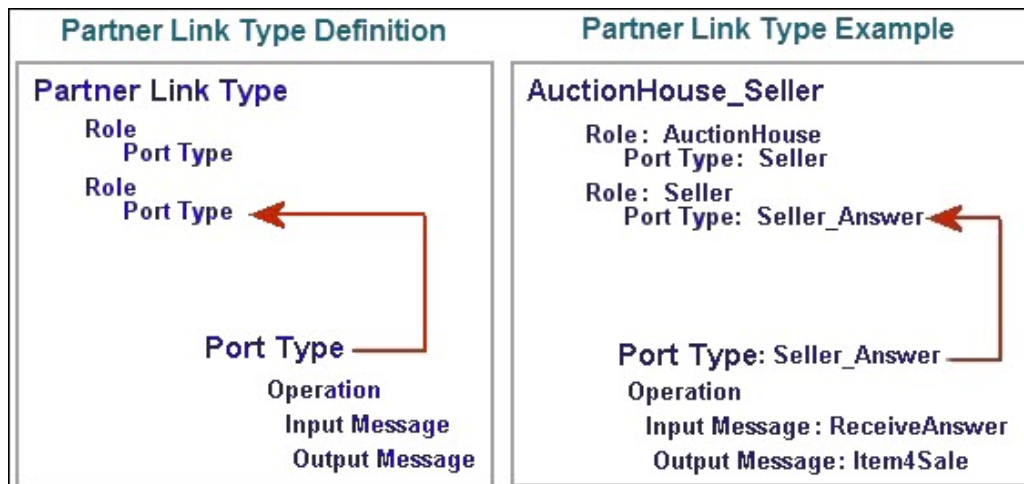
For example, a partner link type named `AuctionHouse_Seller` describes two roles: `AuctionHouse` and *Seller*. The `AuctionHouse` role supports a port type of *Seller* that expects an input message of an item for sale. The `Seller` role supports a port type `Seller_Answer` that expects an input message from the auction house, regarding whether or not the item has sold. The output message is the name of the item for sale.

In this example, the port types are from the same service; however, port types can be from different services.

The partner link type is a WSDL extension. It can specify one or two roles. The port types can be from the same or from different WSDL files.

A partner link type definition can come from the following sources:

- From a WSDL file where it is already defined, or where you can add a new definition.
- From a separate WSDL file with its own namespace for the case where there are two port types, and they are from different services.
- From a Process Developer BPEL process, where you create can a new definition and add it to a WSDL file.

You can add a new partner link type to a WSDL in the following ways:

- From a Project Explorer WSDL
- From a Service Reference
- From an Interfaces View WSDL

These techniques are described in:

- *Adding a new Partner Link Type from a WSDL in Project Explorer*
- *Adding a new Partner Link Type to a new WSDL using a Service Reference WSDL*
- *Add a new Partner Link Type from Interfaces View*

Adding a new Partner Link Type from a WSDL in Project Explorer

Use the following procedure to add a new partner link type from a WSDL in the project explorer:

1. Be sure that you have already imported a WSDL into an orchestration project. Typically, you import WSDL files into the standard `wsdl` folder.
2. Expand the WSDL to show the port type.
3. Right-mouse click on the port type and select *Add to PartnerLink Type*.
4. Complete the wizard that appears. For details, see Step 6 in *Add a new Partner Link Type from Interfaces View*.

Adding a new Partner Link Type to a new WSDL using a Service Reference WSDL
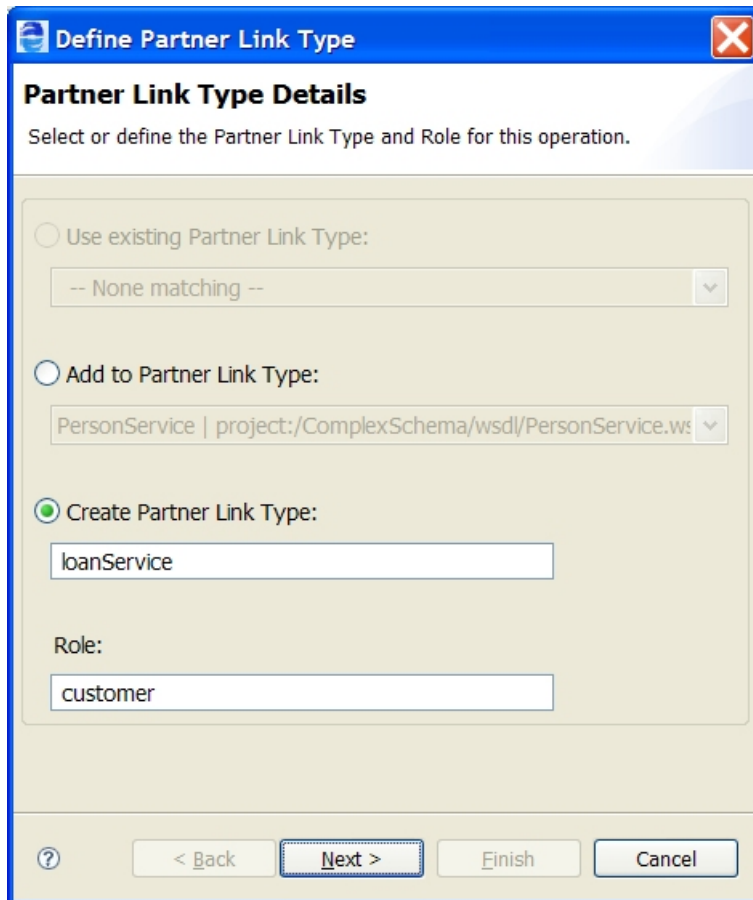
Use the following procedure to add a new parner link type to a new WSDL using a service reference WSDL:

1. Be sure that you have already imported a WSDL into the Service References folder of an orchestration project. This WSDL is a reference to a service already running on a server.

2. Expand the WSDL to show the port type.

3. Right-mouse click on the port type and select **Add to PartnerLink Type**.

4. Complete the wizard that appears. Note that you cannot add a partner link type to the existing remote WSDL. The wizard allows you to create a new WSDL that imports the service reference.

Add a new Partner Link Type from Interfaces View

Use the following procedure to add a new partner link type from *Interfaces* view:

1. Display the Interfaces view. This view is not part of the default perspective. To open it, select **Window > Show View > Interfaces**.

2. Ensure that the port type from the WSDL file you want to refer to is displayed in the list.

3. Expand a *Port Type* to display an operation you want to use, but does not have an associated Partner Link Type yet.

4. *BPEL activity* should be selected.

5. Drag the *operation* to the Process Editor canvas to open the Partner Link Type wizard, as shown.

6. Do one of the following

    - Select *Use an existing Partner Link Type* to select from the list, and then click **Finish**.

    - Select a Partner Link Type definition from *Add to Partner Link Type*, and add a second role (with its associated port type) to the current partner link type. A partner link type can have two roles; they represent two different services that communicate asynchronously. The picklist includes partner link types from Interfaces that currently have only one role. Type in a new role name in the *Role* text box, then click **Finish**.

    - Select *Create Partner Link Type* for a new Partner Link Type name and *Role* name, and then click **Next**.

7. If you created a new Partner Link Type in Step 5, you can add it to an existing WSDL file or create a new file for the definition. Do one of the following:

    - Type in an URL in the *Create new WSDL file* field to qualify all names and message types for this Partner Link Type definition. Then enter the WSDL filename and pathname in the *Location* field, as the example shows in the following illustration, or click **Browse** to create a new filename and location. Type the file name in the *File name* field of the **Open** dialog.

    - Select a WSDL file from *Add to WSDL file* to add your new Partner Link Type definition.
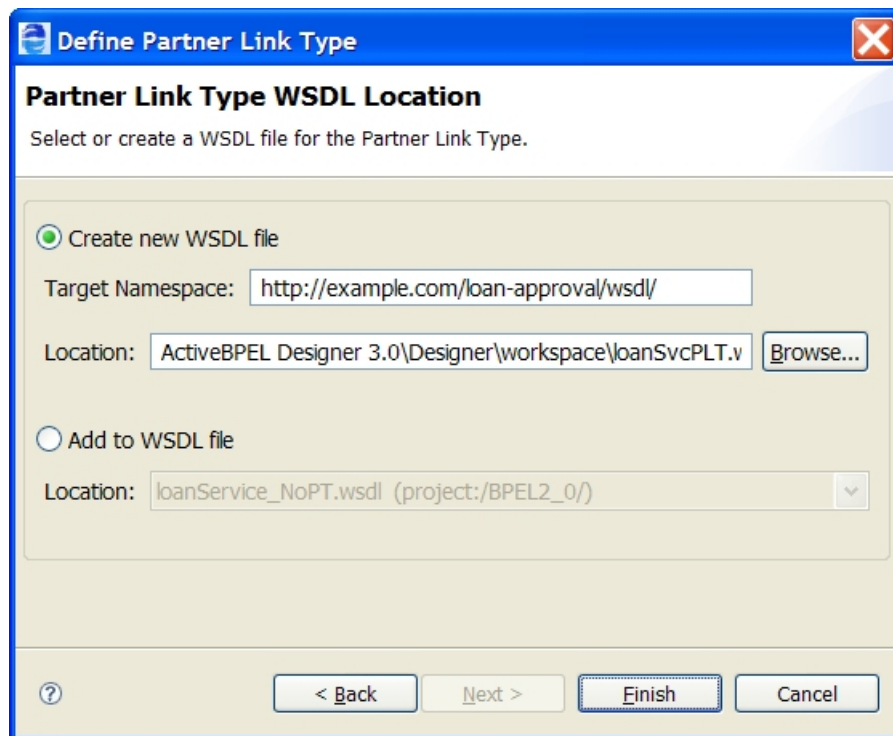
8. Click **Finish**.

If you add the Partner Link Type definitions to a new WSDL, Process Developer automatically adds the WSDL to Project Explorer and to the Imports node in Outline view.

Partner Link

For an easy way to create and use partner link types and partner links, see the key Process Developer feature, *What are Participants?*.

A partner link is a communication exchange between two partners. In the most basic form, the process is a partner link of an external service, receiving a request from it. A partner link defines the role that the process plays (if any) and the role that the partner service plays (if any) in the particular exchange. A partner link can be declared globally or within a scope.

A partner link definition consists of:

- A name
- A partner link type
- At least one role and can include two roles
- Role names of MyRole, which is the process role, and PartnerRole
- Optional property called Initialize Partner Role
- Optional properties for *Adding Comments to a Process* and *Adding Documentation to a Process*
- Optional properties for *Declaring Extension Elements and Attributes*

In a BPEL process, you use the partner link when defining Web service interaction activities. For example, a receive activity specifies a partner link, its port type and operation. The port type comes from the associated partner link type. This means that the process can accept incoming message data from the service playing the partner role defined in the partner link.

At a minimum, a process must have one partner link for each particular communication exchange it is engaged in. BPEL allows for multiple partner links of the same partner link type to exist within a process.

By default, a partner link is declared at the process level. You can declare a partner link at a scope level. The partner link name must be unique among the set of partner links defined by the process or scope.

TheInitialize Partner Role property is provided as a hint to the deployer of a process. This property does not affect the runtime behavior of the process. If the process contains logic to initialize its partner links then it should set this property to No. This type of initialization would occur through the use of an assign activity. If the process requires its deployment environment to initialize its partner links, either through a static endpoint at deployment or perhaps through an endpoint reference scheme like WS-Addressing, then it should set this property to Yes. If the property is missing, then there is no information conveyed to the deployer as to how the partner links are initialized.

Add a partner link automatically to the process:

Partner links are automatically added to the process when you drop a WSDL operation onto the Process Editor canvas and complete a wizard to create a receive, invoke, reply, onMessage, or onEvent. If you must create a new operation, you can use the Operation Wizard. These are the recommended techniques for creating partner links. For more information, see *Creating an Activity by Starting with a WSDL Interface*.

Add a new partner link manually to the process:

1. From the Outline view, right-mouse click on **Participant Partner Link** and select **Add Process Service Consumer** or **Partner Service Provider**.

2. For further details, see *Creating a New Partner Service Interface*.

XML Syntax

```
<partnerLinks>
  <partnerLink name="NCName" partnerLinkType="QName"
    myRole="NCName"? partnerRole="NCName"?
    initializePartnerRole="yes|no"?>+
  </partnerLink>
</partnerLinks>
```

Example

```
<partnerLink name="seller"
  partnerLinkType="AuctionHouse_SellerLT"
```

```
        myRole="AuctionHouse" partnerRole="Seller"
</partnerLink>
```

Using Scoped Partner Links

You can declare a partner link at the process level or at a scope level. While the partner link names may be identical, they are considered unique. Declaring partner links at the process level makes them globally available, except where there is an identically named partner link at the scope level. Scope-level partner links are only available in the scope where they are declared. For details, see *Scope*.

Declaring a partner link at the scope level is useful for a parallel `forEach` activity, in which multiple instances of a scope execute at the same time.

Scoped partner links are also useful for code modularization and reuse. For more information on reusing code in different BPEL processes, see *Creating a Custom Activity*.

Partner Links and Endpoint References

In a BPEL process, interactions between business partners are defined by partner links. The definitions are abstract and do not contain any access information about actual running services. The reason you would omit real access information is to allow the process definition to be reused for many services.

How do the actual services get accessed? A service is identified in two ways: outside of the process at deployment time or within the process in an assign activity.

When a process is deployed, each role defined in a partner link is assigned an endpoint reference. An *endpoint* indicates a specific location for accessing a Web service using a specific protocol and data format. An *endpoint reference* conveys the information needed to access a Web service endpoint. Using endpoint references in deployment makes possible the dynamic selection of a service partner.

Within a process, you can invoke a service by setting a target endpoint reference in an assign activity. For more information, see *Assign*. Also, a partner link can be declared within the process or within a scope activity. For more information, see *Scope*.

CHAPTER 7

Implementing a BPMN Task or Event in BPEL

An activity is a step in a BPEL process. It can describe an interaction with a Web service or it can perform internal process functions. The process must contain at least one activity.

In BPMN, you implement both tasks and events as BPEL activities. However, there is a distinction between a task and the event in which a task performs work and an event is a trigger or result.

BPMN-to-BPEL Implementation of Tasks and Events

In many cases, a BPMN element can be implemented as one or more BPEL activities, described in the tables below, which are the following BPMN types: tasks, throw events, and catch events.

For additional implementations, see *BPMN-to-BPEL Implementation of Gateways and Control Flow*.

Tasks

BPMN Task	BPEL Implementations
Service	Invoke
User	People activity, task type
Script	Assign
Rule	Invoke
Suspend	Suspend
Validate	Validate
Abstract	Empty
Manual	- Empty - People activity, task type

BPMN Task	BPEL Implementations
Send	- Invoke - Reply - People activity, notification type
Receive	Receive

Throw Events

BPMN Throw Event	BPEL Implementation
None/Start/End	- Empty - Assign
Message	- Reply - Invoke - People activity, notification type
Error	- Throw - Rethrow
Compensate	- Compensate - Compensate Scope
Terminate	- Exit - Break, implemented in a `forEach`

Catch Events

Any catch event can be dropped onto an activity's boundary to create a boundary event, or into a scope (event subprocess) to create a handler.

BPMN Catch Event	BPEL Implementation
Timer	- Wait - OnAlarm for pick or event handler
Message	- Receive - OnMessage for pick - OnEvent for event handler
Error	- Catch - Catch all
Compensation	Scope compensation handler
Cancel	Scope termination handler

Overview of BPEL Activities

In the default (BPMN) Process Developer editing style, basic activities are organized according to BPMN types: activities and events. In BPEL, there is no distinction: both activities and events are considered activities.

The following table provides brief definitions and links to more details.

Activity Name	Description
Invoke	Directs a Web service to perform an operation
Assign	Manipulates process variables and partner link endpoint references by creating Copy From/To operations for them
Empty	An activity that does nothing when it executes. Useful for situations where you need an activity but do not want anything to really happen, for example, suppressing a fault by having the empty be the child of the catch.
Suspen	Suspends the process. Useful in a catch or catchAll event to catch unexpected errors.
Validate	Validates the values of variables against their associated XML and WSDL data definition
Opaque	(For Process Developer Classic only). For abstract processes. Stands as a placeholder for an activity that would be used in an executable process.
Receive	Accepts message data from a service partner. Optionally begins a process by creating an instance of the process.
Reply	Sends a response to a partner identified in a matching Receive activity
Signal	Signals are similar to a throw, differing in that there isn't a fault. A signal throws information that is received by a signal that waits for this information.
Throw	Signals a fault. Specifies a standard or custom fault
Rethrow	Passes the fault that was originally caught by the immediately enclosing fault handler to the parent scope
Exit	Stops an executable process immediately, resulting in a faulted process.
Wait	Stops process execution for a specified amount of time or until a deadline is reached
Compensate Scope	Starts compensation on a specified inner scope that has already completed successfully
Compensate	Executes the compensation handler on a named scope or executes the default compensation if no scope is named
Break	Breaks out of a scope or a loop in a while or for each activity. Processing continues normally with the next activity.
Continue	For BPEL-Centric palette only.) Continues with the next iteration in a loop (while, repeat until, or for each). A convenience activity that simplifies processing without the need of specifying all the conditions that may be required to continue normal processing when a certain condition is met.
Start/End/None	Equivalent to the BPEL empty or assign activity, described above.

For information on structured activities, see *Different Ways of Structuring Activities*.

Defining an Activity and Its Properties

You can select and drag any palette item to the Process Editor canvas.

With the activity in focus on the Process Editor canvas, you can set property values in the Properties view. Be sure to select the activity, as shown on the right, and not the bordered sequence, as shown on the left.
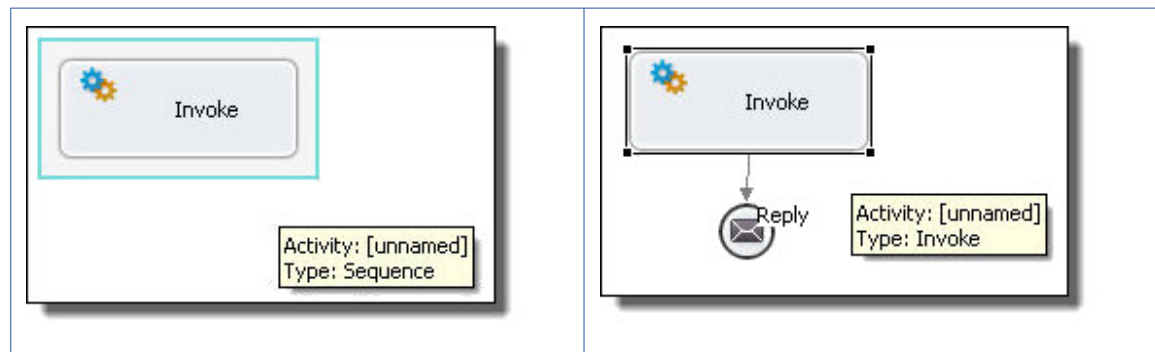
This illustration shows an example of the basic activity properties. The example activity is a reply.

Note that the BPMN construct selected is a Message throw event implemented as a Reply.
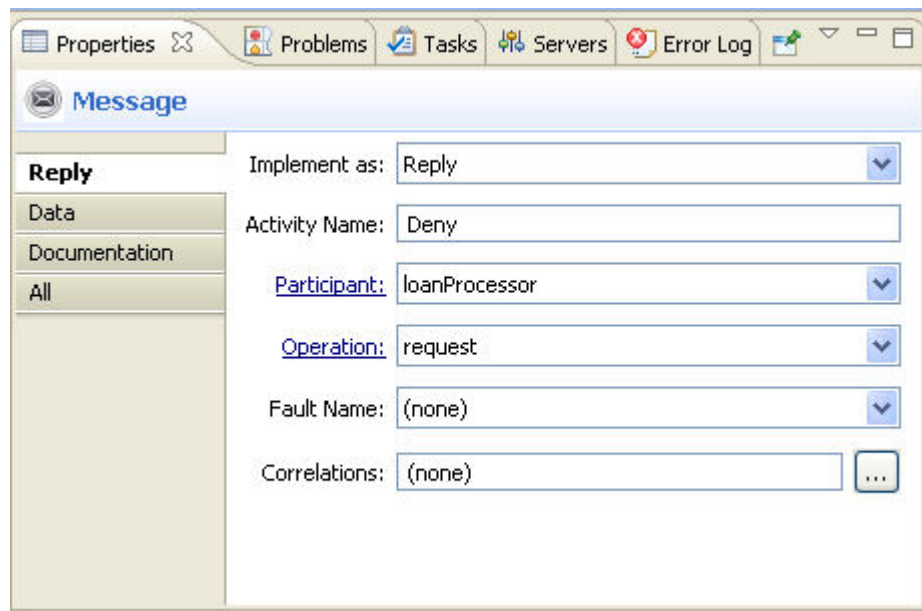
Select the All tab in the Properties view to display all the properties for an activity. The All tab contains the standard properties for all activities. In the illustration below, basic and advanced properties are displayed.
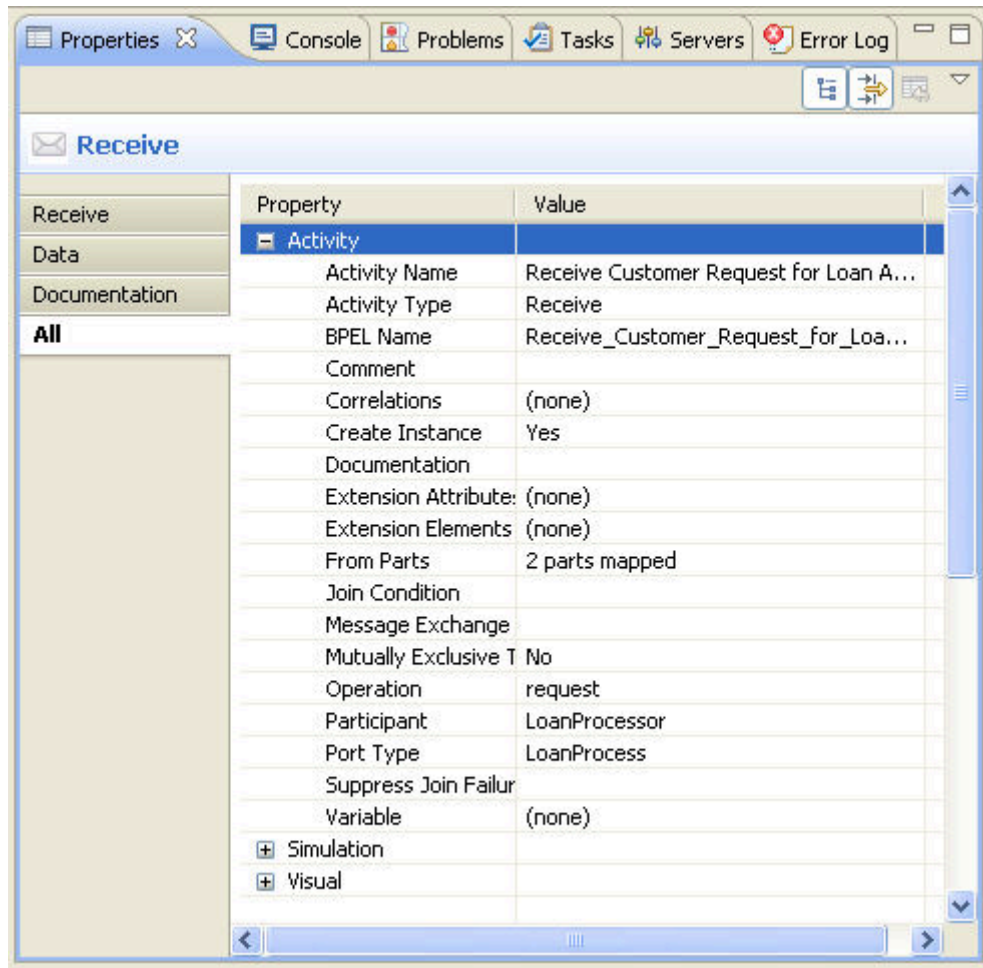
When your mouse hovers over an activity icon, you see property details for the activity. The following illustration shows an example.

For more information, see:

- *Selecting Values for Activity Properties*
- *Selecting Activity Label*
- *Standard Properties for Activities*

Selecting Values for Activity Properties

When you add an activity to the Process Editor canvas, the Properties view displays all of the valid properties for it. Some properties are activity-specific, and some are standard for all activities.

For example, the Create Instance property is valid only for a receive or pick activity. The Copy Operation is valid only for the assign activity.

You assign values to properties as follows:

- For Activity Name, there are several choices. See *Selecting Activity Labels*.

- For list-valued properties, such as Variable, select from the list.

- For building a complex property, such as for Correlation Sets, click the `Dialog (...) Button`.

Selecting Activity Labels

When you add an activity to the Process Editor canvas, it is labeled with the activity type. For example, a receive activity is labeled *Receive*. You can make the label more meaningful by adding a name or description. You can also add a name but display only the type to preserve screen real estate.

The following examples show some labeling choices. Notice that you can keep the label short, if desired, and then use hover help to display details.
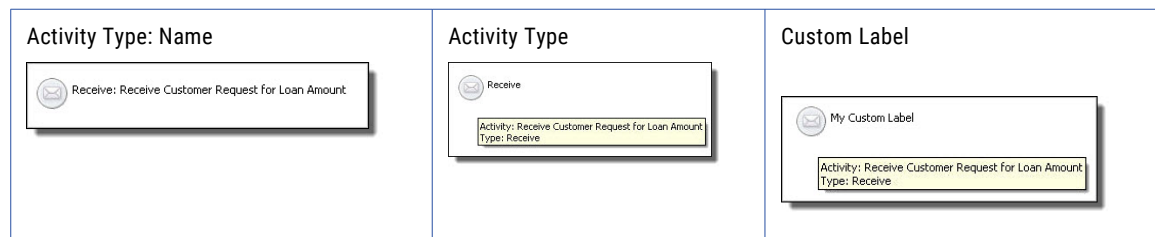
Create a label in one of the following ways:

- Edit the activity label directly on the Process Editor canvas or fill in Activity Name in the Properties view

- Select a label type from the Label property

To edit the activity label:

1. In the Process Editor canvas, click in the activity label. The label field changes to an edit field, as shown.

2. Type in a label that is the same as an activity name. You can include spaces or special characters.

Select the label type from the *Label* field in Properties view as follows:

Activity Name or Type	By default, the type is displayed. If you add a name, the name is displayed and is editable on the canvas.
Activity Name	This is for display purposes only. It can include spaces and special characters, and is editable on the canvas. It does not export with the BPEL XML.
Activity Type	The best choice to preserve screen real estate. The Type is not editable on the canvas.

Activity Type: Name	Meaningful label, but is not editable on the canvas.
Custom	Type text in the Custom Label field

Standard Properties for Activities

The following table shows standard properties for all activities. The table does not include activity-specific properties, which are discussed under each activity.

Attribute	Description
Suppress Join Failure	See *Process Properties*
Join Condition	See *Creating a Join Condition for an Incoming Link*
Comment	See *Adding Comments to a Process*
Documentation	See *Adding Documentation to a Process*
Execution State	See *Viewing the Execution State of an Activity or Link*
Extension Attributes and Elements	See *Declaring Extension Elements and Attributes*

The Suppress Join Failure, Join Condition, and Extension properties appear in the Properties view when you select the All tab on the Properties view.

Adding a Background Color to a Task Scope People Activity and Handlers

To enhance a diagram's meaning, you can add color to the following activities:

- Any bordered activity. Bordered activities are in the **Tasks** drawer of the BPMN palette.
- **Collapsed Scope**. Collapse a scope by right-mouse clicking on it and selecting Collapse Container. A scope is referred to as an embedded subprocess in the BPMN palette, in the Control Flow drawer.
- Collapsed fault, event or compensation handlers

Understanding and Using Activity Sequences and Flows

Each activity that you add to the Process Editor canvas is contained by a hidden sequence activity. The sequence is represented as a blue border when your mouse hovers near it, as shown.
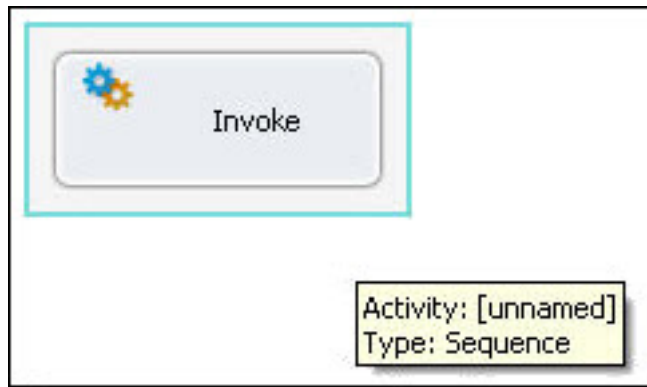
The purpose of hidden sequences is to make it easier for you to link activities together. Activities are automatically linked in sequence when you place one activity near the top, bottom, left, or right of another activity.
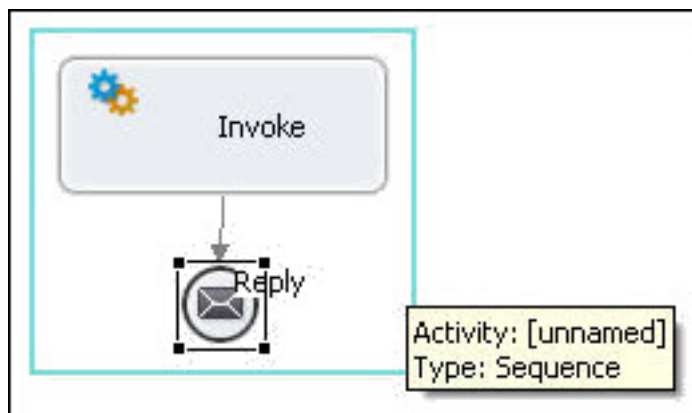
Receive

BPMN Implementation: Receive task, Message catch event

When a BPEL processing engine gets a message, it searches for a receive (or pick) activity with a matching partner link and operation. For an executable process, the receive must specify an input variable or variable part for the message data received. See *Participants* for descriptions of concepts important to this activity.

The receive activity can begin a business process instance by including a *Create Instance* property set to Yes. A receive activity can be associated with a reply activity if the operation is a request-response. Additionally, the receive can include a message exchange attribute.

You can create a set of concurrent initial receives. This case allows for any one message of a set to initialize a BPEL process. To create a multiple-activity starting point, add the receives to a flow container. Also, you must provide a correlation set for all concurrent receives that initiate a process, as described in *Adding Correlations to an Activity*.

Two other activities are similar to a receive: an onMessage clause in a pick activity or and onEvent event handler. For more information, refer to *Pick* and *Event Handling*.

Required Properties	Optional Properties
Participant (Partner Link)	Name. See *Selecting Activity Labels*
Operation	Port Type
Variable (required only for executable process, not abstract process) or From Part to Variable. See *From Part to Variable*.	Correlations. See *Correlation*
	Create Instance. Required if this is the start activity.
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See Process Propertieson page 108
	Comment. See Adding Comments to a Processon page 64
	Documentation. See Adding Documentation to a Processon page 65
	Setting Visual Properties and Using Your Own Library of Imageson page 61
	Execution State. *Viewing the Execution State of an Activity or Link*
	Message Exchange. *Message Exchange Declaration*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To add a receive activity to the process manually:

For a shortcut and recommended technique, see *Creating an Activity by Starting with a WSDL Interface*.

1. Drag a **Receive task** or **Message catch event** activity to the Process Editor canvas.
   You can add a background color to the receive task, but not the message catch event.
2. In the Properties view, select the following values:
   a. Optionally type in a Name.
   b. In the Participant drop-down, select **New Process Service Consumer**.
   c. Select an Operation from the picklist.
3. In the Data tab, do one of the following:
   - Select Single Variable from the Assignment Type and create a new variable or select a process variable.
   - Create a Parts to Variables specification. For details, see *From Part to Variable*.
   - Create an XPaths specification. For details, see *Input Variable*.
4. Select other optional properties as desired.

A receive activity is actually two activities: a Receive followed by an Assign in a Scope. These two activities are not atomic and if you, for example, initialize onAlarm in an event handler, you cannot use the variable until the assign has completed.

Usage Examples

One receive can start a process	In a pick (event-driven gateway) with onMessage activities, one of many can start a process; the rest are ignored	Any one starts a process; all are required to complete the fork join (flow)
Receive: Receive Customer Request for Loan Amount		
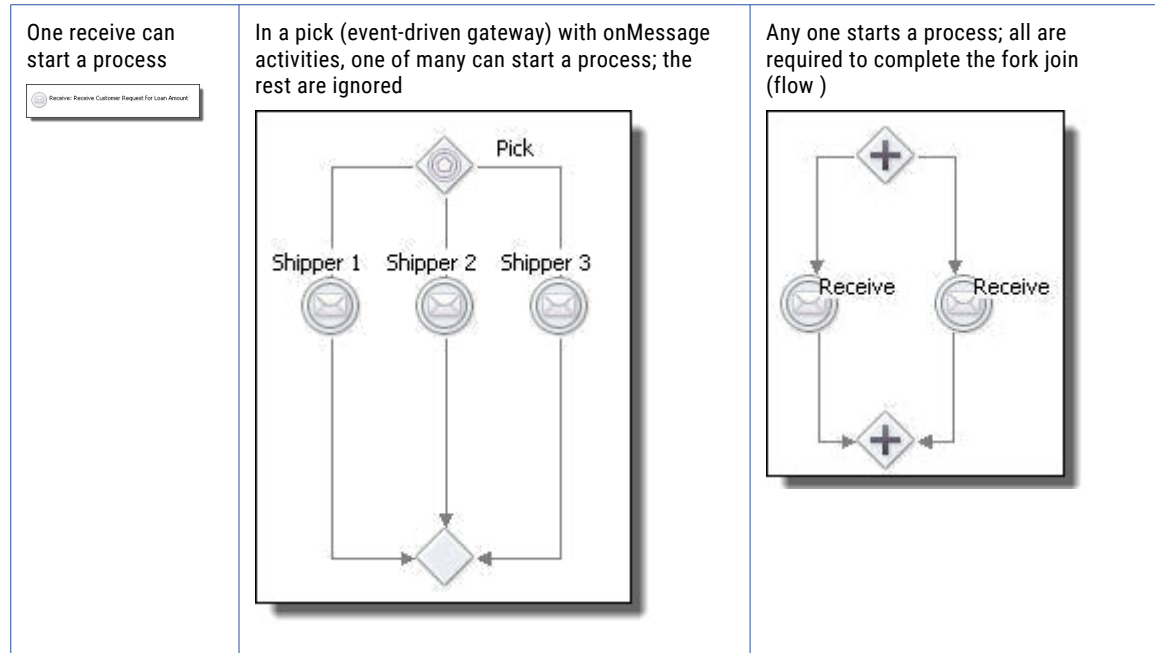
XML Syntax

```
<receive partnerLink="NCName" portType="QName"?
        operation="NCName"
        variable="BPELVariableName"? createInstance="yes|no"?
        messageExchange="NCName"?
        standard-attributes>
    standard-elements
    <correlations>?
        <correlation set="NCName" initiate="yes|join|no"?>+
    </correlations>
    <fromParts>?
        <fromPart part="NCName" toVariable="BPELVariableName"/>+
    </fromParts>
</receive>
```

XML Example:

```
<receive name="ReceiveCustomerRequest"
        partnerLink="customer"
        portType="lns:loanServicePT"
        operation="request"
        variable="request"
        createInstance="yes"/>
```

Reply

BPMN Implementation: Send task, Message throw event

A reply activity sends a message back to a Web service in response to a message received or through fault handling. The received message is in a receive activity, an onMessage handler, an onEvent handler, or an invoke. For a request-response operation, a reply activity has the same partner link and operation as its matching receive, onMessage, or onEvent.

A receive may have multiple reply activities referencing it but only one of them is allowed to execute. For example, there may be a normal reply and a fault-handling reply. Both replies share a partner link and operation with receive, but only one of them is allowed to execute.

For an executable process, a reply activity must specify a variable for the message data being sent.

A reply activity can return a fault response. You can select a reply to handle a fault by adding it to a catch container and specifying the fault name and variable properties. See *Adding a Fault Handler* for more information.

A reply activity can return a fault response. You can select a reply to handle a fault by adding it to a catch container and specifying the fault name and variable properties.

Required Properties	Optional Properties
Participant (Partner Link)	Name. See *Selecting Activity Labels*
Operation	Port Type
Variable See *Input Variable* or toPart fromVariable See *From Variable to Part*.	*Correlation*
	Fault Name
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Message Exchange. See *Message Exchange Declaration*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To add a reply activity to the process manually:

For a shortcut and recommended technique, see *Creating an Activity by Starting with a WSDL Interface*.

1. Drag a **Send task** or **Message throw event** to the Process Editor canvas.
   You can add a background color to the send task, but not the message throw event.

2. In the Properties view, select the following values:

    a. Optionally type in a Name.

    b. In the Participant drop-down, select **New Process Service Consumer**. See *Creating a New Process Service Consumer Interface*.

    c. Select an **Operation** from the picklist.

3. In the Data tab, do one of the following:

    *   Select Single Variable from the Assignment Type and select a variable, or select New Variable.

    *   Select XPaths or XQuery. For details, see *Input Variable* and *From Part to Variable*.

4. Select other optional properties as desired.

A simple example of using a reply activity in a process is shown in the following illustration.
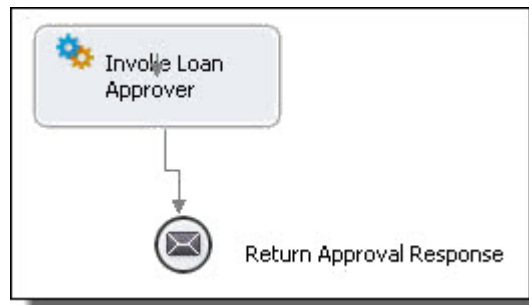
To add a reply activity with a fault response:

1. Create a catch container for the appropriate scope or the process as a whole. For more information, see *Defining Catch and CatchAll Fault Handlers*.

2. Create a catch container for the appropriate scope or the process as a whole.

3. Add a reply to the catch container.

4. Select the properties and name, as described in the procedure above.

5. Select a Fault Name from the list. The fault name is defined in a WSDL file.

6. Select the fault variable from the Variable list.

The following illustration shows an example of a reply with fault. The fault name is shown in the main tab. The variable is specified in the Data tab.
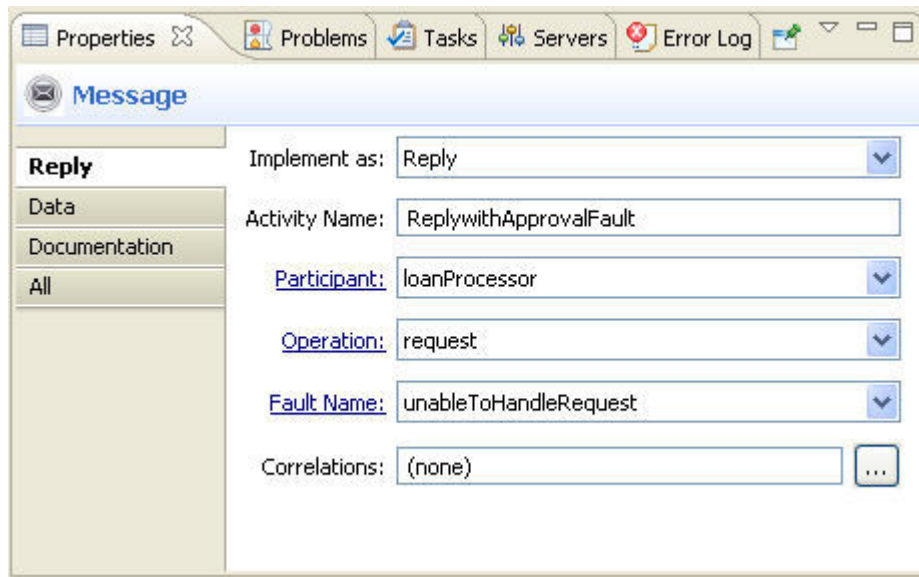
The following illustration shows a simple example of using a reply with a fault in a catch handler.

XML Syntax

```
<reply partnerLink="NCName" portType="QName"?
        operation="NCName"
        variable="BPELVariableName"? faultName="QName"?
        messageExchange="NCName"?
        standard-attributes>
  standard-elements
  <correlations>?
      <correlation set="NCName" initiate="yes|join|no"?>+
  </correlations>
  <toParts>?
      <toPart part="NCName" fromVariable="BPELVariableName"/>+
  <toParts>
</reply>
```

Example 1 Normal Response:

```
<reply name="reply" partnerLink="customer"
        portType="lns:loanServicePT" operation="request"
        variable="approval">
```

Example 2 Fault Handling:

```
<reply name="reply" partnerLink="customer"
        portType="lns:loanServicePT" operation="request"
        variable="error" faultName="unableToHandleRequest">
```

Throw

BPMN Implementation: Error throw event

The throw activity provides one way to handle errors in a BPEL process by generating a fault. You can set link conditions from an activity and then connect one link to a throw. For a discussion of error handling, see *Fault Handling*.

The throw activity provides one way to handle errors in a BPEL process by generating a fault. You can set link conditions from an activity and then connect one link to a throw.

For a list of faults described in the BPEL specification, see *BPEL Standard Faults* elsewhere in this help.

Required Properties	Optional Properties
Fault Name	Name. See *Selecting Activity Labels*
	Fault Variable
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a throw activity:

1. Drag an **Error throw event** to the Process Editor canvas.
2. In Properties view, specify a fault to throw.

The following illustration shows an example of using a throw activity.
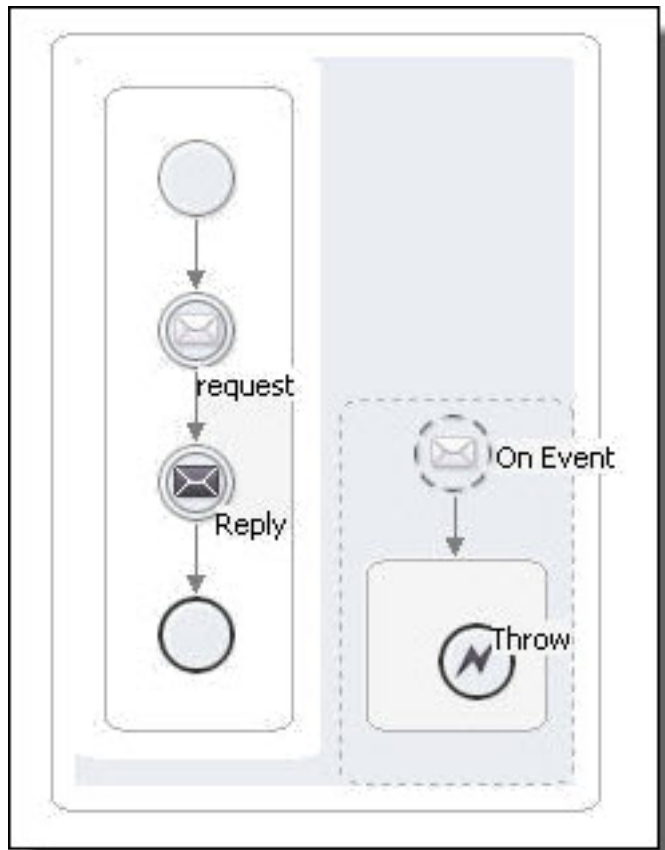
XML Syntax

```
<throw faultName="QName"
          faultVariable="BPELVariableName"? standard-attributes
      standard-elements
</throw>
```

Example:

```
<throw xmlns:FLT="http://example.com/faults"
          faultName="FLT:OutofStock"/>
```

Rethrow

BPMN Implementation: Error throw event

The rethrow activity passes to the parent scope the fault that is originally caught by the immediately enclosing fault handler. The rethrow activity can be used only within a fault handler's catch and catch all elements. The activity always throws the original fault data and preserves its type.

A rethrow activity is part of a default catch all fault handler. See *Fault Handling* for more information.

A rethrow activity is part of a default catch all fault handler.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To add a rethrow activity to the process:

1. On the Process Editor's Fault Handler tab or a scope fault handler, add a **catch** or **catch all** container.

2. Drag an **Error throw event** to a catch or catch all container.

The following illustration shows a simple example of using a rethrow activity within a catchAll fault handler of a scope.
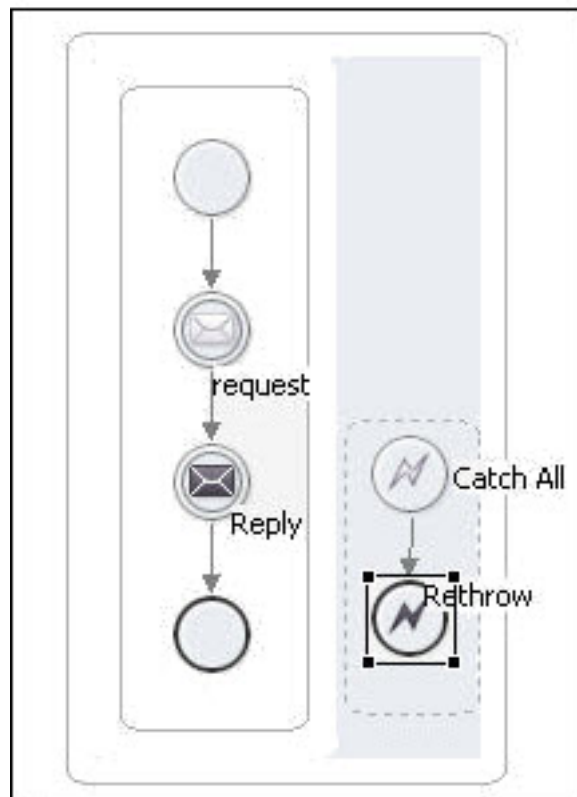
Tip: if you collapse fault handler, you can add a background color to it in the Properties view.

XML Syntax

```
<rethrow standard-attributes
         standard-elements
</rethrow>
```

Example:

```
<rethrow/>
```

Signal

A signal is an event that you create that throws information within a process and which can be received anywhere within the process. Catching a signal event is equivalent to catching a message or alarm event except that only the name of the signal needs to be specified. This means it could be caught inline (like a wait), as an alternative in a pick (like onAlarm), as a handler (like onAlarm), or using a boundary event.

Note: This is a Process Developer extension activity.

Unlike a message (which a signal resembles), a signal can be received by more than one activity. It also differs in that a message must know the activity to which the message is directed. In contrast, a catch signal is broadcast and the signal decides which of the signals being broadcast it should process.

To add signals to your process, drag a Signal icon to the Process Editor canvas. Signal resides in the Throw and Catch event palettes.
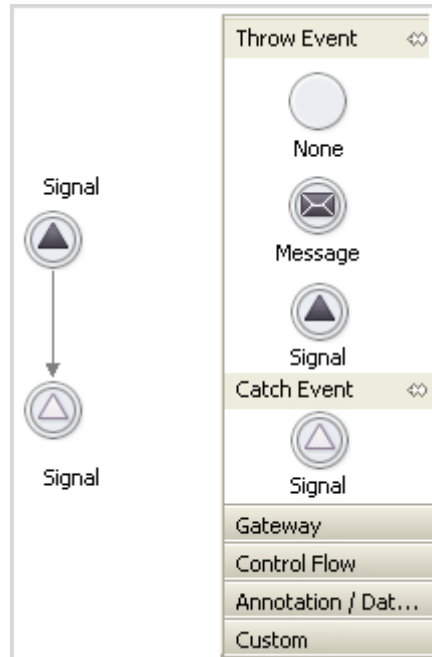
After dragging a throw signal to the canvas, you must assign it a *Signal Name*. Do not confuse this with an *Activity Name*. When you define the properties of the catch signal, you will identify the signal it catches by this name. It needs this name because a process can have more than one signal.

When a signal is thrown, all signal handlers that are active and waiting a signal of that name execute and they do this before the end of the current transaction.

For information on defining a signal's properties, see *Defining an Activity and Its Properties*.

If you are using an existing workspace and have customized your palette, signal events may not show in it. If you run into this problem, copy the signal events entries over into the workspace or remove the workspace's customized palette. This does not occur when you create a new workspace signal events show in the palette.

Exit

BPMN Implementation: Terminate throw event

The exit activity stops a business process. It is designed for an executable process, not an abstract process. All currently running activities must be terminated immediately without any fault handling or compensation.

Termination of a process with an exit activity is an abnormal completion. The process ends with a Process Developer system fault, `processTerminated`.

Required Properties	Optional Properties
none	Name.
	Join Condition.
	Suppress Join Failure.
	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images
	Execution State.
	Extension Attributes and Extension Elements.

To build an exit activity:

1. Drag a **Terminate throw event** to the Process Editor canvas.

2. Add the activity to an appropriate container or link another activity to it.

The following illustration shows an example of using an exit activity.
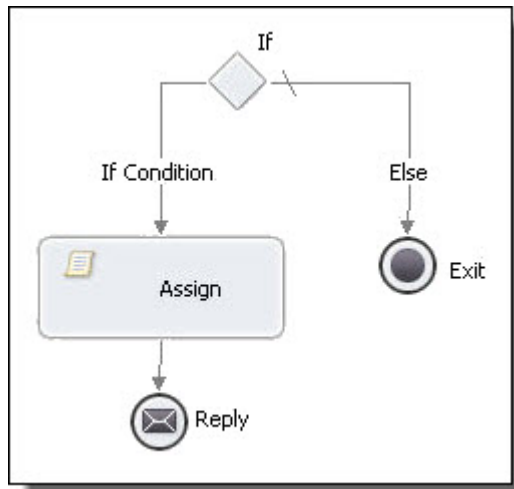
XML Syntax

```
<exit standard-attributes
        standard-elements
</exit>
```

Wait

BPMN Implementation: Timer catch event

The wait activity tells the business process to wait for a given time period or until a certain time has passed.

The wait expression conforms to an expression, where the value is an XML schema type of dateTime, date, or duration.

Required Properties	Optional Properties
Wait Expression	*Name*
Wait Type	*Join Condition*
	Suppress Join Failure
	Comment
	Documentation
	Setting Visual Properties and Using Your Own Library of Images
	Execution State
	Extension Elements and Attributes.

To build a wait activity

1. Drag a **Timer catch event** to the Process Editor canvas.

2. In the Properties view, select a Wait Type: Duration or Deadline. You can select a static duration or deadline or create an expression.

3. Select the duration or deadline static control or click the `Dialog (...) Button` next to Deadline or Duration Expression.

4. In the Expression box, specify one of the following:

   - For a deadline, specify a date and time, such as `'2010-12-12T12:00'`. Be sure to include the single quotes.

   - For a duration, specify a time, such as `'P1DT10S'`. Be sure to include the single quotes.

In addition to the few examples below for deadline and duration expressions, you can see the examples given in the XML schema specification.

The following illustration shows an example of using a wait activity.

See the following:

- `http://www.w3.org/TR/xmlschema-2/#duration`

- `http://www.w3.org/TR/xmlschema-2/#datetime`

- `http://www.w3.org/TR/xmlschema-2/#date`
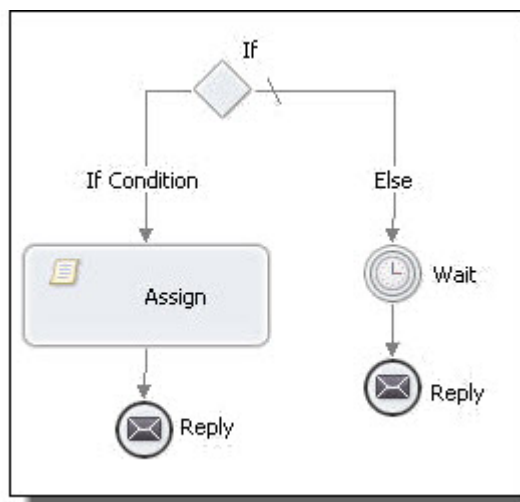
XML Syntax

```
<wait (for="duration-expr" |
      until="deadline-expr")standard-attributes>
    standard-elements
</wait>
```

Examples:

- A duration of one second:

  ```
  <wait for="'PT1S'"/>
  ```

- A duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes:

  ```
  <wait for= "'P1Y2M3DT10H30M' "/>
  ```

- A deadline with a date and time:

  ```
  <wait until="'2010-12-12T12:00'"/>
  ```

Compensate

BPMN Implementation: Compensate throw event

The compensate activity triggers the start of compensation on all inner scopes that have already completed successfully. See *Compensation* for a discussion of compensation handlers.

The compensate activity specifies no scope name, thus providing explicit default-order compensation that executes the compensation handler on all eligible scopes in the reverse order of their completion.

The compensate activity is defined within a compensation handler, fault handler, or termination handler. It can also be the target of a compensation boundary event.

The compensate activity is one of two compensation activities. The *Compensate Scope* activity names an enclosed scope for compensation.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a compensate activity:

1. On the Process Editor canvas, select a scope that has a compensation, fault, or termination handler container.
    For a fault handler, ensure that the fault handler contains a catch container.

2. Drag a **Compensate throw event** to the container, as shown in the following example.
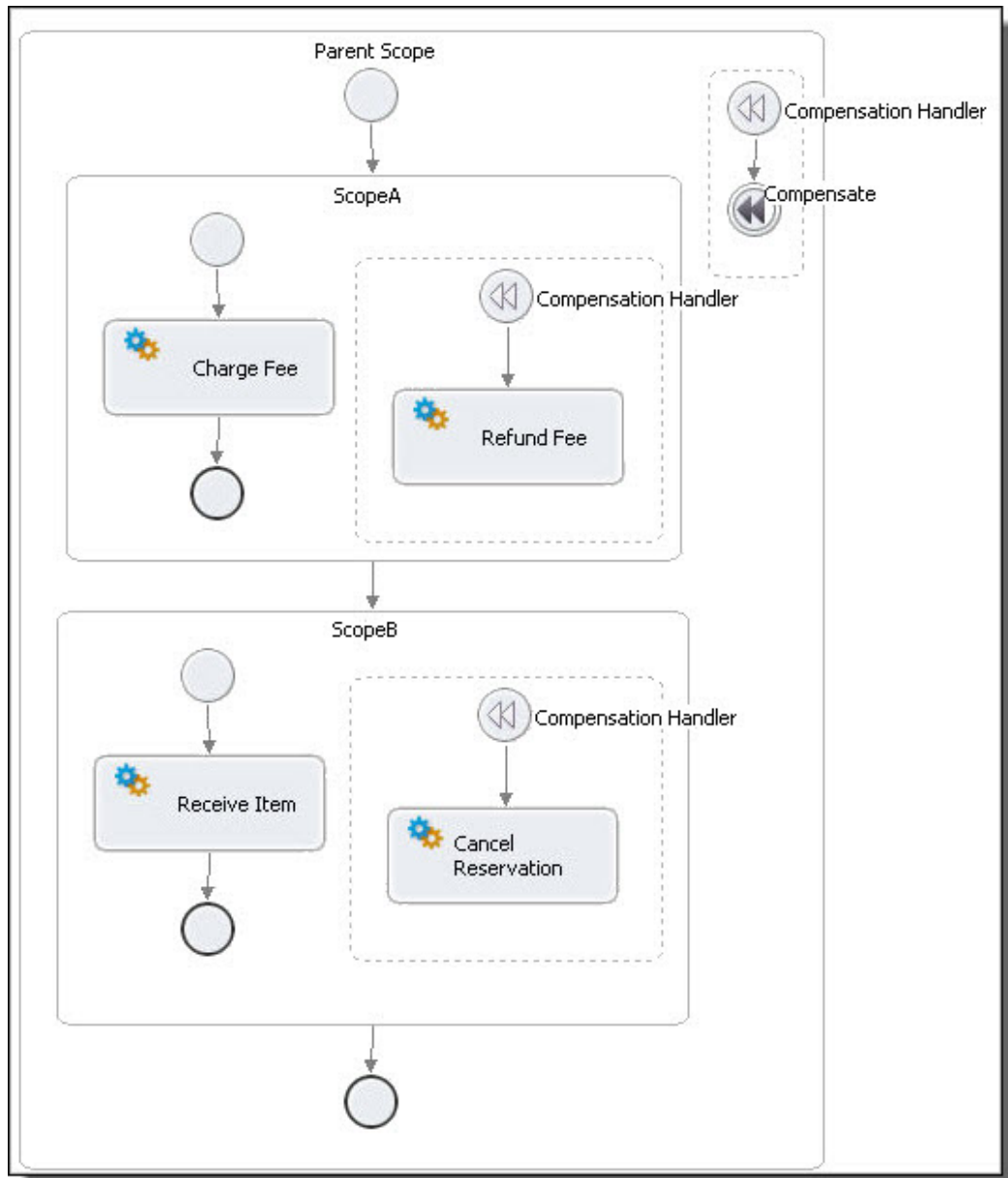
Alternately, use a BPMN compensation boundary event and make the compensate activity a target of it. For details, see *Adding Boundary Events* and *Catch and Catch All Boundary Events and Compensate, Compensate Scope and Rethrow*.

XML Syntax

```
<compensate standard-attributes>
        standard-elements
</compensate>
```

Example

```
<compensate/>
```

Compensate Scope

BPMN Implementation: Compensate throw event

The compensate scope activity names an enclosed scope for compensation and triggers the start of the scope's compensation handler. See *Compensation* for a discussion of compensation handlers.

The compensate scope activity is one of two compensation activities. The compensate activity does not name an enclosed scope, but rather executes the compensation handler on all eligible enclosed scopes in the reverse order of their completion.

Required Properties	Optional Properties
Target	Name. Se
	Join Condition.
	Suppress Join Failure.
	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images
	Execution State
	Extension Attributes and Extension Elements.

To build a compensate scope activity:

1. On the Process Editor canvas, select a scope that has a compensation, fault, or termination handler. For a fault handler, ensure that the fault handler contains a catch or catch all container.
2. Drag a **Compensate throw event** to a catch, catch all, compensation handler, or termination handler, as shown in the following example.
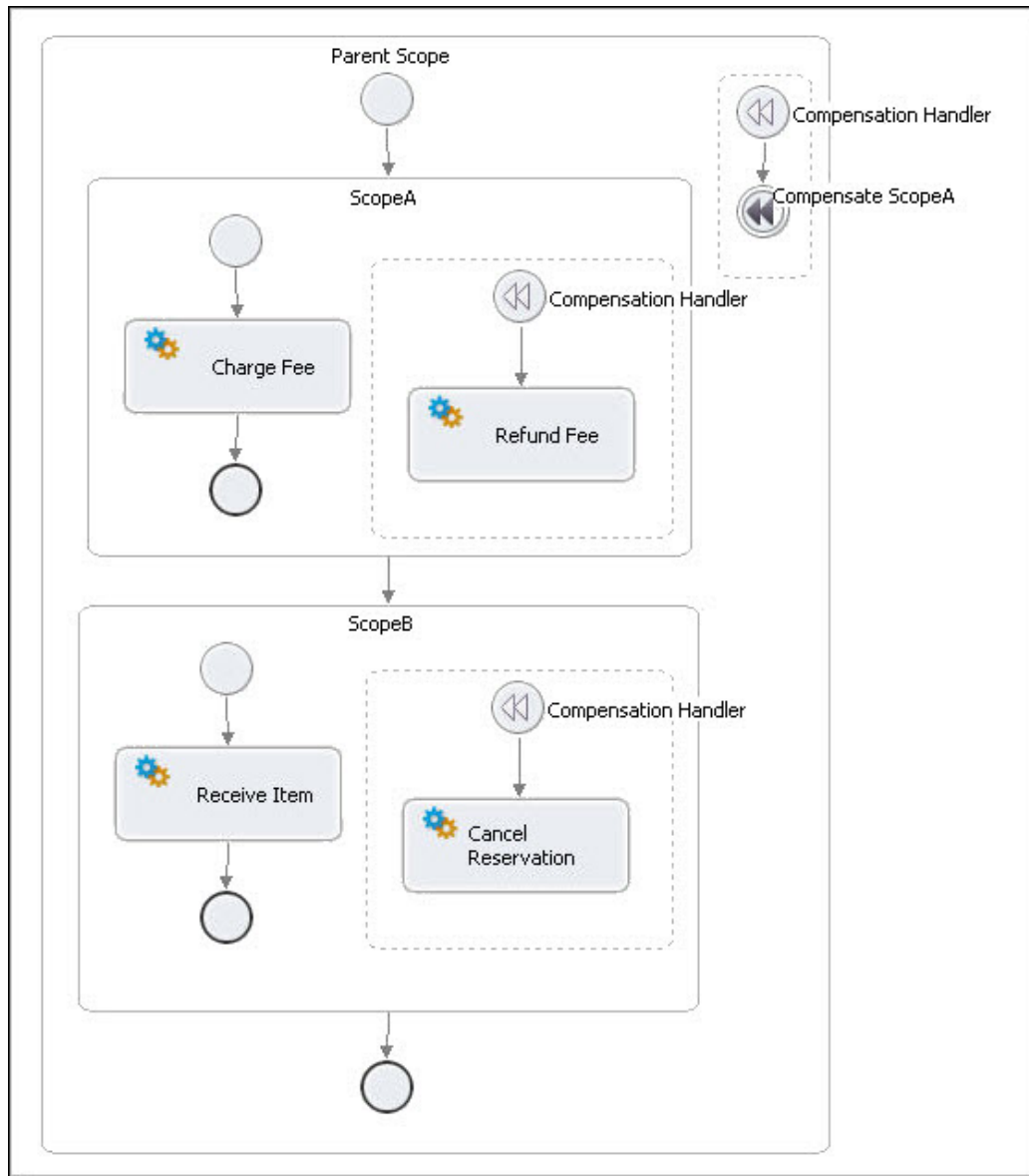
In the Properties view, select the arrow in the Target row, and select a scope name from the list. The scope must be an inner scope in relation to the compensate scope activity.

If no scope names are available, you can not add a compensate scope activity to the current scope.

XML Syntax

```
<compensateScope target="NCName" standard-attributes>
        standard-elements
</compensateScope>
```

Example

```
<compensateScope target="assessorScope"/>
```

Break

BPMN Implementation: Terminate throw event

This is a Process Developer extension activity. When you save the process, a message appears informing you that the process contains an activity that is not in the BPEL specification.

By default, the break activity exits the nearest enclosing scope, while, or `forEach` activity. You can use a break inside a container to break out of processing.

When a break executes, control is transferred to the target of the break activity. The target does not execute any further iterations and completes normally. When multiple scope, while, or `forEach` activities are nested within each other, a break activity applies only to the innermost activity. In the case of a parallel `forEach`, the break activity causes all of the executing iterations to terminate all of their executing child activities and complete normally. The break activity allows the root scope of a `forEach` to complete normally so that it is compensatable. Early termination through a break activity is not considered a faulting state, and therefore does not prevent compensation.

Note that you can set the BPMN Terminate property to No to change the behavior of the break to break out of loops (while, for Each and repeat until), but not scopes.

You can use a break to exit normally instead of throwing a fault and forcing termination of the process.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition.
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.
	BPMN Terminate. If set to Yes (the default), stops execution out to the nearest scope, while loop or for-each loop, whichever comes first. Repeat-until loops are skipped, since they are not represented with a boundary (like a scope) and BPMN terminate does not see it as a loop. The scope or loop is considered to have completed successfully (same as throwing a fault and then catching it). If set to No, breaks out of loops (while, for Each and repeat until), but not scopes.

To add a break activity:

1. On the Process Editor canvas, select a scope, while, or `forEach` container.

2. Drag a **Terminate throw event** to it.

The following illustration shows an example.
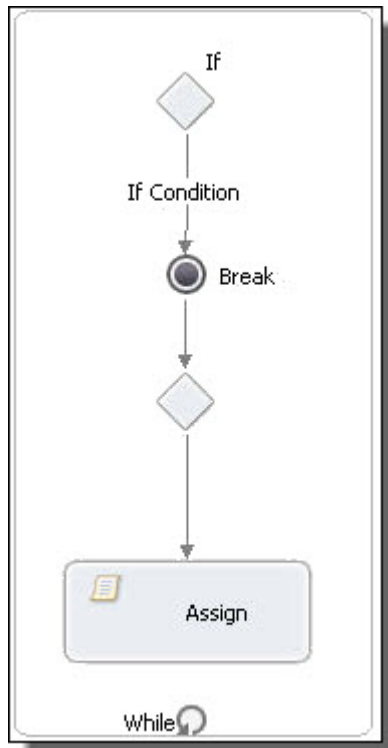
XML Syntax

```
<break standard-attributes>
   standard-elements
</break>
```

Example

```
<while>
   <targets><target linkName="Link2"/></targets>
   <sources><source linkName="Link5"/></sources>
   <condition>($counter &lt; 5)</condition>
   <flow>
     <links>
       <link name="Link5"/>
     </links>
     <if>
        <sources><source linkName="Link5"/></sources>
        <condition>($counter = 3)</condition>
       <extensionActivity>
        ext:break name="Break"/>
       </extensionActivity>
     </if>
     <assign>
        <targets><target linkName="Link5"/></targets>
        <copy>
            <from>($counter + 1)</from>
            <to variable="counter"/>
        </copy>
     </assign>
   </flow>
</while>
```

Continue

BPMN Implementation: None

This is a Process Developer extension activity. When you save the process, a message appears informing you that the process contains an activity that is not in the BPEL specification.

You can use the continue activity inside a while, repeat until, or `forEach` container. The continue activity causes its immediately enclosed looping activity to stop the execution of the current iteration and move on to the next iteration. In the case of a while, the while's condition expression is evaluated to see if another iteration is possible. In the case of a serial `forEach`, the next iteration in the sequence executes or the loop completes if the current iteration is the final one. In the case of a parallel `forEach`, the continue only affects its enclosed iteration, allowing any parallel iterations to complete normally.

The continue activity allows the root scope of a `forEach` to complete normally so that it is compensatable. Early termination through a continue activity is not considered a faulting state, and therefore does not prevent compensation.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To add a continue activity:

You must use the BPEL-Centric palette to use the continue activity.

1. On the Process Editor canvas, select a while, repeat until, or `forEach` container.
2. From the **Event** palette, drag a continue activity to it.

The following illustration shows an example.
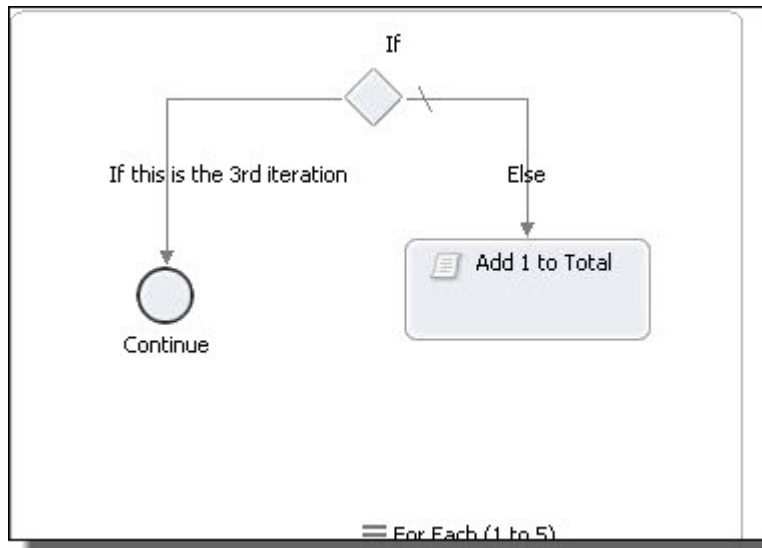
XML Syntax

```
<continue standard-attributes>
    standard-elements
</continue>
```

Start End None

BPEL Implementation: Empty or Assign

The none events are useful at the beginning and end of a control flow to indicate the boundary of the flow. The start event does not have a trigger. The end event does not have a result.

The intermediate event can be added within a control flow, such as the else if branch of an If activity.

Invoke

BPMN Implementation: Send task, Service task, Rule task, Message throw event

The invoke activity directs a Web service to perform a one-way or request-response operation. It specifies the participant that provides the service and the operation to invoke. The invoke activity must specify the data for the message transmitted and may specify an output variable or variable part in the case of a synchronous request-response Web service invocation. See *Participants* for descriptions of concepts important to this activity.

An invoke activity can be compensated, or reversed. For more information, see *Compensation*.

An invoke activity can also include fault handling when the invoke is added to a scope. Alternately, any faults associated with the invoked service can be handled at the process level, if desired.

An invoke activity can also include fault handling when the invoke is added to a scope. Alternately, any faults associated with the invoked service can be handled at the process level, if desired. For more information, see *Fault Handling*.

Required Properties	Optional Properties
Participant (Partner Link)	Name. See *Selecting Activity Labels*
Operation	Port Type
Input Variable See *Input Variable*. or toPart fromVariable See *From Variable to Part*.	Correlations. See *Correlation*.
	Output Variable See *Output Variable* and From *Part to Variable*.
	Input and Output assignments
	Join Condition. *See Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To add an invoke activity to the process manually:

1. Drag a **Send task** (or other BPMN implementation mentioned above) to the Process Editor canvas.

2. You can add a background color to the send, service, and rule tasks.
   In the Properties view, select the following values:

   a. Optionally type in a Name.

   b. In the Participant drop-down, select **New Partner Service Provider**. See *Creating a New Partner Service Interface*.

   c. Select a participant's Operation from the picklist.

3. In the Input tab, do one of the following:

   • Select Single Variable from the **Assignment Type** and select a variable.

   • Select a **XPaths** or **XQuery.** For details, see *From Part to Variable*.

4. Optionally, select an *output variable* or a *to part*. For details, see *From Variable to Part*.

5. Select other optional properties as desired.

A simple example of using an invoke activity in a process is shown in the following illustration.
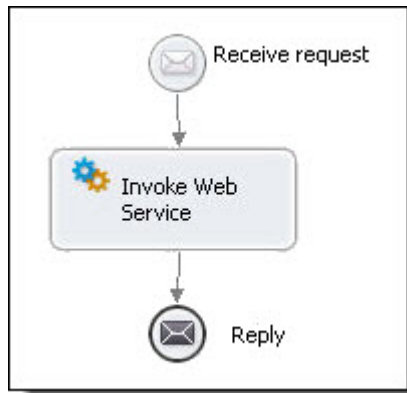
For another shortcut for creating an invoke activity, see *Creating an Activity by Starting with a WSDL Interface*.

XML Syntax

```
<invoke partnerLink="NCName" portType="QName"? operation="NCName"
      inputVariable="NCName"? outputVariable="NCName"?
      standard-attributes>
  standard-elements
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"?
        pattern="request|response|request-response"/>+
  </correlations>
  <catch faultName="QName"? faultVariable="NCName"?
      faultMessageType="QName"? faultElement="QName"?>*
      activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
  <compensationHandler>?
      activity
  </compensationHandler>
  <toParts>?
      <toPart part="NCName" fromVariable="BPELVariableName"/>+
  </toParts>
  <fromParts>?
      <fromPart part="NCName" toVariable="BPELVariableName"/>+
  </fromParts>
</invoke>
```

Example:

```
<invoke name="invokeapprover" partnerLink="approver"
      portType="lns:loanApprovalPT" operation="approve"
      inputVariable="request" outputVariable="approval">
```

From Part to Variable

Select a variable part and assign it to a process variable. Select New to create a From Part or To Part specification to add to the list.

`Receive`, `onMessage` (pick activity), `onEvent` (event handler), `invoke`, and people activities can copy message parts to process variables, eliminating the need to create an assign activity to do the copy. You can specify a from pan instead of a variable for a receive, `onMessage`, and `onEvent` activity. You can also specify a from pan in place of the output variable for an invoke or people activity.

If the WSDL operation uses a message containing exactly one part which itself is defined using an element, then the Web service interaction activity can use the fromPart element for the activity's variable. The fromPart declarations are used in place of the activity's variable declaration. In the case of an invoke or people activity, the fromPart declaration replaces the output variable.

Select the part and variable from the list of in-scope variables. For an onEvent activity, specify a variable name, and the variable is created implicitly.

The following example show the XML source for a From Part assignment.

```
<onEvent operation="asyncOp" partnerLink="requestPLT">
    <fromParts>
        <fromPart part="one" toVariable="myNewVariable"/>
    </fromParts>
    <scope/>
</onEvent>
```

From Variable to Part

Select New to create a From Part or To Part specification to add to the list. Or select a process variable and assign it to a variable part

Reply and invoke activities can copy a process variable to a message part, eliminating the need to create an assign activity to do the copy. For the invoke, you can specify a to part instead of an input variable, and for the reply, you can specify a to part instead of the variable.

If the WSDL operation uses a message containing exactly one part which itself is defined using an element, then the Web service interaction activity can use the toPart element for the output variable. The toPart declarations are used in place of the activity's output variable declaration.

The following example shows the XML source for a To Part assignment.

```
<invoke outputVariable="twoWayRequest"
    name="InvokeWithManagedCorrelation" operation="asyncOp"
    partnerLink="requestPLT" portType="ns1:asyncRequestPT">
    <toParts>
        <toPart fromVariable="V1" part="one"/>
    </toParts>
</invoke>
```

Input Variable

The input for an invoke, reply, or people activity can be constructed within the Input (or Data) tab of the activity's *Properties* view. Use the Input (or Data) tab of the invoke, reply or people activity as follows:

The input for an invoke or reply activity can be constructed within the Input (or Data) tab of the activity's *Properties* view. Use the Input (or Data) tab of the invoke or reply as follows:

Assignment Type

- **Single Variable**. Select a variable if no input preparation is desired via an assign activity. Select an existing process variable or select New Variable. The variable must be of the operation's input message type, or, if the operation requires only one part, the variable's type can be the element type of the one part.

- **XPaths**. Select XPaths if you want to use a table to map from data within existing variables or a literal to the contents of the input message. The table uses XPath expressions that allow you to select nodes from a variable to assign to the input variable as well as compute values (strings, numbers, or Boolean values) from the content of variable. See **Adding XPaths** below.

- **XQuery**. Select XQuery if you want to fill each input message part with a single query written in the XQuery expression language. This language allows you to create a query that looks like an XML document, but where pieces of the document are constructed using a superset of the XPath expression syntax. It supplements XPath with a SQL-like "FLWOR expression" (FOR, LET, WHERE, ORDER BY, and RETURN) for performing joins. See **Adding XQuery** below.

For example XPath and XQuery expressions, see *Selecting XPath or XQuery for Expression Building*.

Tip: If you want to use Javascript to prepare some or all of an input message, create an assign activity instead of using the Input tab. Each copy operation in an assign can use a different expression language.

Adding XPaths

1. On the Input (or Data) tab of an invoke, reply, or people activity, ensure that the Assignment Type is XPaths.

2. On the Input (or Data) tab of an invoke or reply activity, ensure that the Assignment Type is XPaths.

3. In the table, select **Add** to populate the b column.

4. In the To Path column, select the child node of the part, if needed.

5. In the E/L column, select the From type: Expression or Literal.

6. In the From column, select the `Dialog (...) Button` at the end of the table cell. For a literal, select **Generate** and notice that the literal XML document is generated for you based on the type needed for the chosen To Part and To Path. To add an expression, fill in the Expression text box. For a shortcut, see *Using Content Assist*.
   For more information on expression languages, see *Using the Expression Builder*.

   The following is an example of a literal mapping:

7. Optionally, in **Copy Attachments**, select a variable that can have attachments that you want to copy to or from. Note that you must copy all attachments. If you want only some attachments, you can add an Assign activity before the activity. See *Adding an Attachment* for details.

8. When you use XPaths, a temporary variable called *parameters* is created to contain the parts. For details, see *Viewing Container Variables* below.

Adding XQuery

For easy XQuery development, see *Writing XQuery Functions*.

1. On the Input (or Data) tab of an invoke, reply, or people activity, ensure that the Assignment Type is XQuery.
   Notice that a XML template is generated for the message part. If desired, after editing the document, you can regenerate the template.

2. Fill in the XQuery expressions as desired for each element. Optionally, select **Builder** to open the Query Builder to insert expressions or functions into the query.
   The following is an example of an XQuery mapping.
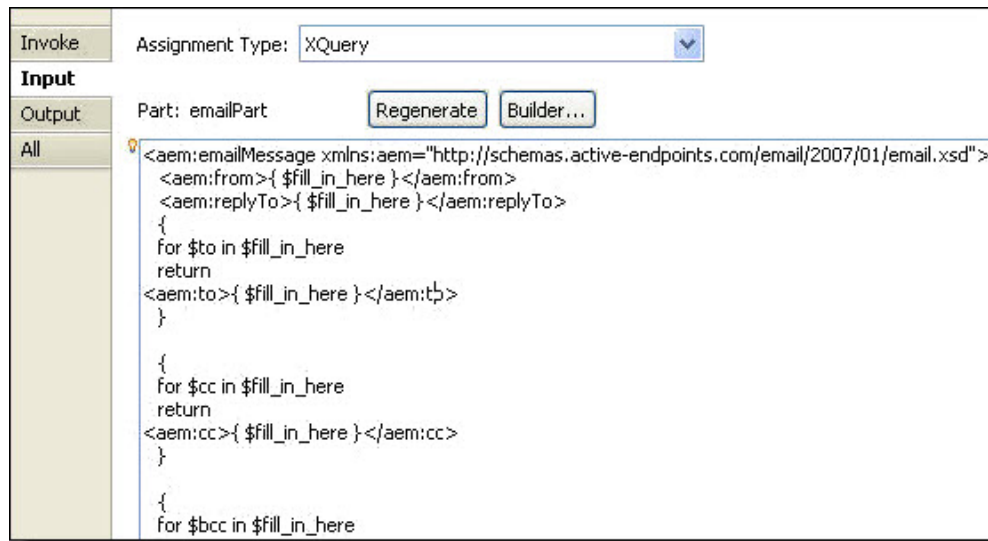
3. Optionally, copy attachments, as described above in the XPaths discussion.

Viewing Container Variables

Iif you use XPaths or XQuery, a container variable is added to the Process Variables view, called `parameters`. Similarly, if you are copying attachments to the input variable, you will see the container variable, `attachmentCopyResult`. These variables are hidden by default since they are used only temporarily at runtime to process parts of a full variable. Select **Show Internal Variables** to make them visible.

Output Variable

The output from a receive, invoke (for a request-response operation) or people activity (containing a task) can require that data is mapped to a process variable.

The output from a receive or invoke (for a request-response operation) can require that data is mapped to a process variable.

Assignment Type

- **Single Variable**. Select a variable from the list to put the output of the activity into that variable. The variable must be of the operation's output message type, or, if the operation reply has only one part, the variable's type can be the element type of the one part. Select an existing process variable or select *New Variable*.

- **Parts To Variables**. Select Parts To Variables if you want each reply message part to map to a different variable. For details, see *From Part to Variable*.

- **XPaths**. Select XPaths if you want to use a table to map from data within existing variables to the contents of the input message. The table uses XPath expressions that allow you to select nodes from a variable to assign to the input variable as well as compute values (strings, numbers, or Boolean values) from the content of variable. See *Adding XPaths* in the *Input Variable* topic.

Viewing Container Variables

If you use XPaths, a container variable is added to the Process Variables view, called `result`. Similarly, if you add are copying attachments to the input variable, you will see the container variable, `attachmentCopyResult`. These variables are hidden by default since they are used only temporarily at runtime to process parts of a full variable. Select **Show Internal Variables** to make them visible.

Assign

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

BPMN Implementation: Script task, None/Start/End throw event

The assign activity updates the content of variables. It updates variables in several ways:

- Copies data from one variable to another for a **Copy** operation
- Executes a script with a **Script** operation without the need for specifying a copy from/to definition
- Constructs new data using XPath (or other language) expressions and functions
- Constructs new data using WS-BPEL and other extension functions

You can also use assign to copy endpoint references to and from partner links for dynamic selection of service partners.

If desired, you can validate the value of variables against their XML or WSDL definitions.

Tip: You can take a shortcut to creating an assign by creating data assignments within a receive, invoke, reply, and people activity. For details, see *Output Variable*.

Required Properties	Optional Properties
Copy optional attributes: Keep Source Element Name. See *Validating Variables* Ignore Missing From Data. See Copy Operation with Ignore Missing From Data Attribute.	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Validate. See *Validating Variables*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To add an assign activity to the process:

1. From the **Task** palette, drag a script task to the Process Editor canvas.
   You can add a background color to the script task.

2. Double-click the assign to open the Operations dialog.
   Alternately, in the Properties view, click in the Value field of the Copy Operations and select the `Dialog (...)` `Button`.

3. Select **New Copy** or **New Script**.

4. For a New Script, the Script Builder opens where you can create an expression to execute. The script operation avoids the need for defining a from/to specification that a copy operation uses.

5. For a New Copy, in the From section, select a **Type**.

From Type	Description	Upon Selection of this Type…
Expression	An XPath (or other language) expression to perform computations on variables and properties. Must return a string, number or Boolean.	You can create the expression with an Expression Builder. For more information, see *Using the Expression Builder*
Literal	A constant that conforms to an XML schema definition. The type can be indicated inline using `xsi:type`	Do one of the following: Type in literal value (for example, 100) Select the Literal Contents `Dialog (...)` `Button` to open a larger editing area Generate literal contents for a complex variable, described in *Using the XML Data Wizard* and *Copy Operation Literal Contents Examples*. Assign a dynamic endpoint reference. See Copy Operation Dynamic Endpoint Reference Example.
Opaque	Name representing private application data. Use this option only for an abstract process. For more information, see *Creating an Executable vs. an Abstract Process*.	No other choices are allowed
Partner	This selection generates a list of partner links and roles	Select a partner link and role defining an endpoint reference
Variable	A WSDL message variable and optional part name A WSDL message property XML schema simple type or element	Select a variable name, and optionally, a part name and query. For more information, see *Using the Query Builder*. Queries are allowed only in executable processes.
Variable Property	A WSDL message property	Select a variable name and property

6. In the To section, select a Type. The To type must be compatible with the From type, as shown.

This From type	Must match this To type
Message-type variable	Message-type variable with matching type
XML schema type variable	A variable containing a schema type
XML element type variable	A variable containing the same element
Literal	Must have a matching XML schema type

This From type	Must match this To type
Expression	Expression or Variable
Partnerlink	partner role

7. If desired, enable the **Keep Source Element Name** check box, applicable only for element-based variables. The source element name is used as the name of the resulting destination element. See *Element to Element Copy Operation with Keep Source Element Name Attribute*.

8. If desired, enable the Ignore Missing From Datacheck box to avoid a selection failure if the source variable is missing optional data. See Copy Operation with Ignore Missing From Data Attribute.

9. Click **OK**. You can add multiple operations to an Assign. They are executed in the order shown in the **Operations** dialog and also in the Outline view.

10. If desired, in the Properties view, set the Validate property to yes to validate all variables used in the assign's Copy operations.

For a shortcut to creating new assigns or adding Copy operations to an existing assign, you can use drag-and-drop features in the Process Variables view. For more information, see *Using Variables in a Copy Operation*.

Copy Operation XML Syntax

```
<assign validate="yes|no"? standard-attributes>
  standard-elements
  (<copy keepSrcElementName="yes|no"?
      ignoreMissingFromData="yes|no"?>
    from-spec (see below)
    to-spec (see below)
  </copy> |
  <extensionAssignOperation>
    ...assign-element-of-other-namespace...
  </extensionAssignOperation>) +
</assign>
```

From-Spec XML Syntax

The From element can be in one of these forms:

```
<from variable="BPELVariableName" part="NCName"?
    <query queryLanguage="anyURI"?>?
      queryContent
    </query>
/from>
<from partnerLink="NCName"
    endpointReference="myRole|partnerRole"/>
<from variable="BPELVariableName" property="QName"/>
<from expressionLanguage="anyURI"?>expression</from>
<from><literal>literal value</literal></from>
```

In an abstract process, you can also use:

```
<from opaque="yes">
```

Opaque data is private application data that you do not want to expose to your business partners. It is effectively an abstract process placeholder for some real data that you intend to use in an executable process.

To-Spec XML Syntax

The To element can be in one of these forms:

```
<to variable="BPELVariableName" part="NCName"?
    <query queryLanguage="anyURI"?>?
      queryContent
```

```
        </query>
/to>
<to partnerLink="NCName"/>
<to variable="BPELVariableName" property="QName"/>
<to expressionLanguage="anyURI"?>expression</to>
```

Examples:

Example 1

```
<assign>
   <copy>
     <from>'yes'</from>
     <to part="accept" variable="approval"/>
   </copy>
</assign>
<assign>
   <copy>
      <from>$po.lineItem[@prodCode=$myProd]/amt * $exchangeRate
      </from>
      <to>$convertedPO.lineItem[@prodCode=$myProd]/amt</to>
   </copy>
</assign>
```

Example 2: XML for a Script Operation

```
<bpel:assign>
    <bpel:extensionAssignOperation>
       <bpel:documentation>new script operation
                    </bpel:documentation>
       <ext3:script>abx:createAttachment('request',
           'text/plain',
           abx:base64Encode("Test attachment") )</ext3:script>
    </bpel:extensionAssignOperation>
</bpel:assign>
```

See also:

- Tips for Copy Operations

- Copy Operation Query and Expression Examples

- Copy Operation Literal Contents Examples

- Copy Operation Dynamic Endpoint Reference Example

- Element to Element Copy Operation with Keep Source Element Name Attribute

- Copy Operation with Ignore Missing From Data Attribute

Tips for Copy Operations

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

Here are a few types when using a Copy operation:

- Copy operations in Assign activities are executed in the order listed in the Operations dialog. The order is also shown in the *Outline* view.

- You can re-order the operations in either the Operations dialog or in the Outline view.

- You can also add a comment or documentation for each copy operation. Select an operation from the Outline view. The properties for it are displayed, and you can add a comment and documentation for it. For more information, see *Adding Comments to a Process*.

Copy Operation Query and Expression Examples

Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity. Select a FROM Type and then complete the Copy operation by making other appropriate selections.

Example 1: Query selection

Example 2: Expression using built-in BPEL function

See "Using the Expression Builder " on page 240 for more information.

Copy Operation Literal Contents Examples

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity. For a complex type or element, you can generate an XML data file. If the Generate button is greyed out, the variable selected is

not of a compatible type. For any variable, use this editor to type in appropriate literal values. Select any matching type or element from the drop-down list to create a XML data file for the root element.

Copy Operation Literal Contents Examples

Type in a literal value for copying to a simple variable, variable property, expression, or partner link, as shown in Example 1. Generate XML data for a complex variable, as shown in Example 2.

Example 1: Literal value

The expression above assigns a string to variable part that is a string. If the variable part were an integer, Process Developer would not report an error, rather it would generate a zero value. Process Developer converts any illegal values to zero for numeric type conversions.

Example 2: Literal contents generation used to initialize a complex variable

Do the following:

1. In the TO panel, select a complex variable (and optionally one of its parts) that you want to generate XML data for.
    If you do not select a TO target first, an unfiltered list of top-level schema elements or complex types that the process is importing are available. If no elements or types are available, the **Generate** button is unavailable.

2. Select the `Dialog (...) Button` below the Literal Contents text box to open a larger edit box.

3. Select **Generate**. The **Generate** button is available only if there are top-level schema elements or complex types in the imported WSDL. The list is filtered if you have selected a known type or element in the TO panel.

4. Fill in the details for generating data, as described in *Using the XML Data Wizard*.

5. Edit details of the generated data in the Literal Contents edit box, if desired, as shown in the example.
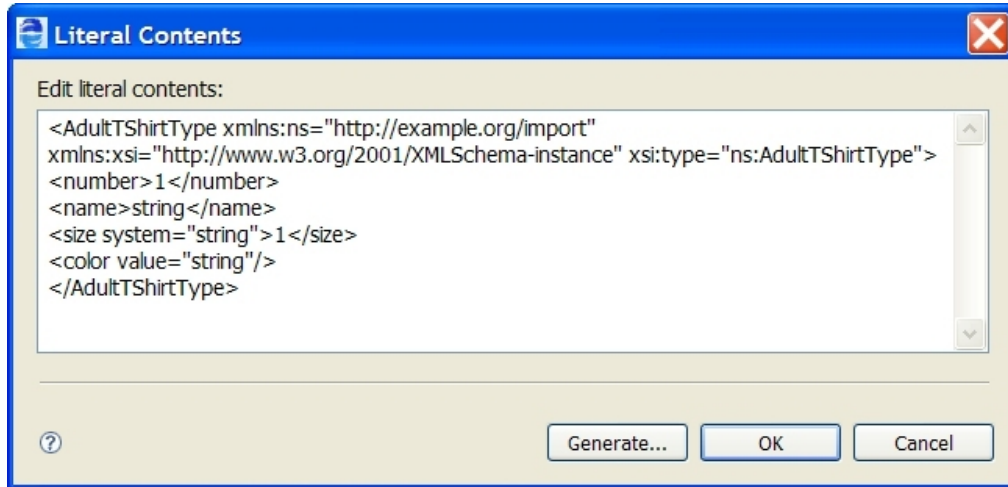
6. Click **OK** to view the copy operation, as shown in the example.
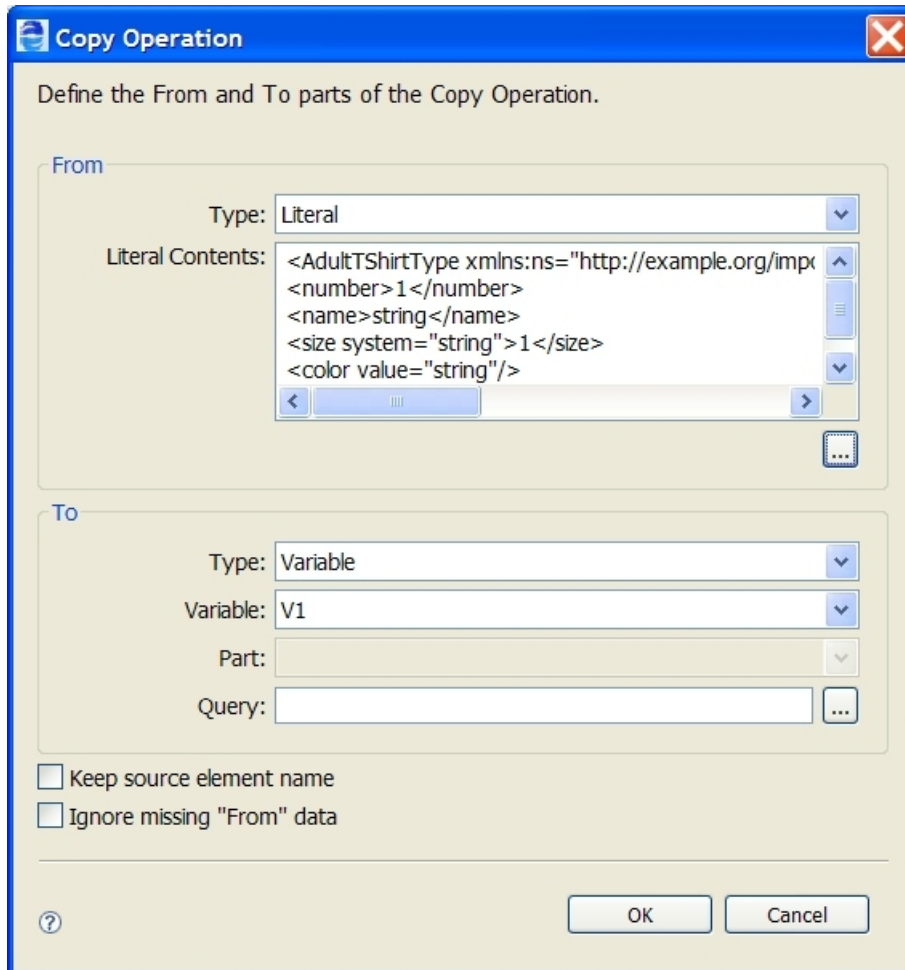
Copy Operation Dynamic Endpoint Reference Example

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

Here is an example of how to specify a copy operation for dynamic assignment of a service. The service name would be supplied as an input to the Assign activity.

Be sure to declare the WS-Addressing namespace when you refer to endpoint references. Process Developer supports several versions of the WS-Addressing specification, as described in *Endpoint References and WS-Addressing Considerations*.

To implement this assignment, type in a valid endpoint reference in the Literal Contents text box. Click the `Dialog (...) Button` below the text box to open a larger edit box, if desired. Copy from a Literal to a Partner Link, as shown:

Example:

```
<assign>
    <copy>
        <from>
            <wsa:EndpointReference
                xmlns:s="http://www.active-endpoints.com/wsdl/bpr-epr"
                xmlns:wsa=
                    "xmlns:wsa="http://www.w3.org/2005/08/addressing">
                <wsa:Address>anyURI</wsa:Address>
                <wsa:ServiceName>tns:Service</wsa:ServiceName>
            </wsa:EndpointReference>
        </from>
        <to partnerLink="mypartnerLink"/>
```

```
            </copy>
        </assign>
```

Element to Element Copy Operation with Keep Source Element Name Attribute

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

Use **Keep Source Element Name** to replace the element at the destination with a copy of the entire element at the source, including children and attribute properties. This attribute supports XSD substitution groups and choice.

If Keep Source Element Name is disabled, then the resulting value for element-based variable in the destination (that is, the TO side) has its original namespace and local name properties.

If Keep Source Element Name is enabled, then the resulting value for the element-based variable in the destination uses the name of the element-based variable in the source (that is, the FROM side).

Given the following schema snippet:

```
<xsd:element name="poHeader">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:choice>
            <xsd:element name="shippingAddr"
               type="tns:AddressType" />
            <xsd:element name="USshippingAddr"
```

```
                type="tns:USAddressType" />
        </xsd:choice>
        <xsd:element name="billingAddr"
                type="tns:AddressType" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Given the following process variable values prior to the copy operation:

poHeaderVar1	poHeaderVar2
<tns:poHeader>	<tns:poHeader>
<tns:USshippingAddr verified="true">	
<tns:street>123 Main Street</tns:street>	
<tns:city>SomeWhere City</tns:city>	<tns:shippingAddr pobox="true" />
<tns:country>USA</tns:country>	
<tns:zipcode>98765</tns:zipcode>	
</tns:USshippingAddr>	
</tns:poHeader>	</tns:poHeader>

Given the following copy operation:

```
<assign>
    <copy keepSrcElementName="yes">
        <from>$poHeaderVar1/tns:USshippingAddr</from>
        <to>$poHeaderVar2/tns:shippingAddr</to>
    </copy>
</assign>
```

The resulting value of `poHeaderVar2` is:

```
<tns:poHeader>
    <tns:USshippingAddr verified="true">
    <tns:street>123 Main Street</tns:street>
    <tns:city>SomeWhere City</tns:city>
    <tns:country>USA</tns:country>
    <tns:zipcode>98765</tns:zipcode>
    </tns:USshippingAddr>
</tns:poHeader>
```

Copy Operation with Ignore Missing From Data Attribute

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

Use **Ignore Missing From Data** to avoid a selection failure when copying data that can not exist in the source variable. If enabled for a copy operation, then any from-spec that selects zero elements, attributes, or text items causes the copy operation to have no effect. The to-spec for the copy operation is not evaluated and nothing changes in the target variable. If this attribute is not enabled, then the regular bpel:selectionFailure fault is raised if the from-spec selects zero items.

Consider the sample below where a customer record could contain special instructions for package delivery. If these instructions exist in the record then they should be copied over to the data sent to the shipping service. If they do not exist, then the default instruction of "NONE" remains in place.

Case 1: Optional Data is Present

Sample data before the copy operation:

```
<CustomerRecord>
    <Id>100256</Id>
    <ShippingAddress>... </ShippingAddress>
    <DeliveryInstructions>
        Please leave at the back door
     </DeliveryInstructions>
</CustomerRecord>
<ShippingRecord>
    ...
    <SpecialHandling>NONE</SpecialHandling>
</ShippingRecord>
```

Copy Operation:

```
<copy ignoreMissingFromData="yes">
    <from>$customerRecord/DeliveryInstructions/text()</from>
    <to>$shippingRecord/SpecialHandling</to>
</copy>
```

Result after copy operation:

```
<ShippingRecord>
    ...
    <SpecialHandling>Please leave at the back door
    </SpecialHandling>
</ShippingRecord>
```

Case 2: Optional Data is Missing

Sample data before the copy operation:

```
<CustomerRecord>
    <Id>100256</Id>
    <ShippingAddress...>... </ShippingAddress>
</CustomerRecord>
<ShippingRecord>
    ...
    <SpecialHandling>NONE</SpecialHandling>
</ShippingRecord>
```

Copy Operation:

```
<copy ignoreMissingFromData="yes">
    <from>$customerRecord/DeliveryInstructions/text()</from>
    <to>$shippingRecord/SpecialHandling</to>
</copy>
```

Result after copy operation:

```
<ShippingRecord>
    ...
    <SpecialHandling>NONE
    </SpecialHandling>
</ShippingRecord>
```

For further details on selection failure settings, see: *Using the Process Developer Disable Selection Failure Fault Extension*.

Empty

BPMN Implementation: Abstract task, Start/End/None throw event

The empty activity is a "no op" instruction in the business process. This activity is useful for synchronization of concurrent activities. For example, you can use an empty activity to show how links are affected, whether or not a fault is caught, and at what scope level it is caught.

Required Properties	Optional Properties
none	Name.
	Join Condition.
	Suppress Join Failure.
	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images
	Execution State.
	Extension Attributes and Extension Elements.

To build an empty activity:

1. Drag an **Abstract task** (or Start/End/None throw event) to the Process Editor canvas.
   You can add a background color to the abstract task, but not the throw event.

2. Add the activity to an appropriate container or link another activity to it.

The following illustration shows an example of using the empty activity, implemented as an abstract task.
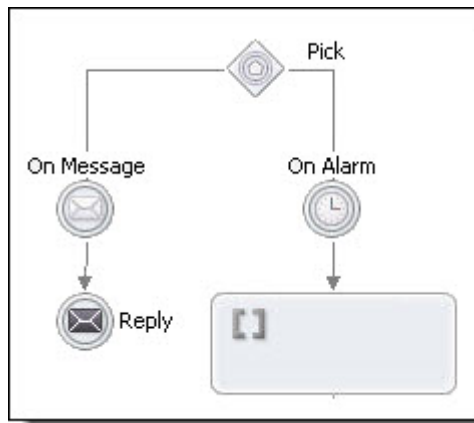
XML Syntax

```
<empty standard-attributes>
     standard-elements
</empty>
```

Suspend

This is a Process Developer extension activity. When you save the process, a message appears informing you that the process contains an activity that is not in the BPEL specification.

The suspend activity suspends a running process. This activity is useful for catching unexpected faults to allow manual intervention for viewing and correcting errors. The optional variable attribute can be used to attach data to an alert which both automatic and programmatic suspends can trigger.

You can perform process exception management on a suspended process. Using the Process Console, you can conveniently view the location of a faulting activity.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Variable
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a suspend activity:

1. Drag a **Suspend task** to the Process Editor canvas.
   You can add a background color to the suspend task.

2. Add the activity to an appropriate container, such as a fault handler's catch or catchAll activity, or link another activity to it.

The following illustration shows an example of using the suspend activity.
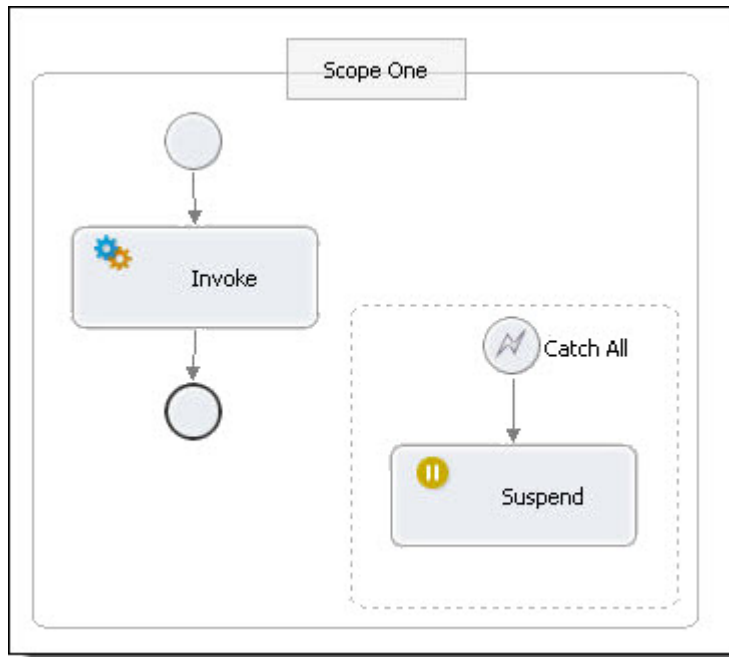
XML Syntax

```
<suspend variable="NCName"? standard-attributes>
    standard-elements
</suspend>
```

Example

```
<catchAll>
    <extensionActivity>
        <ext:suspend/>
    </extensionActivity>
</catchAll>
```

Validate

Select variables for the validate activity's variable list.

The validate activity validates the values of process variables against their associated XML and WSDL data definitions. You can add a list of variables to this activity for validation, regardless of whether or how they are used within a BPEL process.

During simulation, if a variable is found to contain an invalid value, the process terminates with a `bpel:invalidVariables` fault.

Note that you can also validate the specific variables used within an assign activity. See *Assign* for details.

Note also that if the Keep Source Element Name attribute is enabled on a copy operation, it can cause the target variable to be invalid.

For an overview of WS-BPEL validation features, see *Validating Variables*.

Required Properties	Optional Properties
Variables	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*

To add a validate activity to the process:

1. Drag a **Validate task** to the Process Editor canvas.
    You can add a background color to the validate task.
2. Double-click the validate to open the Select Variable dialog.
    Alternately, in the Properties view, click in the Variables field and click Dialog (...).
3. In the **Select Variable** dialog, select one or more process variables for validation against their XML or WSDL definition.

The following illustration shows a simple example of using a validate activity to check the values of variables after an assign activity.
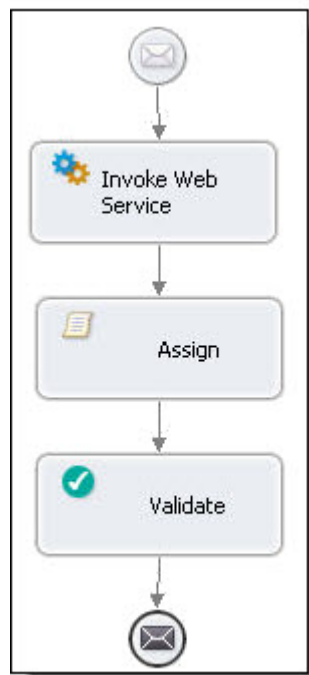
XML Syntax

```
<validate variables="BPELVariableName-list"
        standard-attributes>
    standard-elements
</validate>
```

Opaque

Process Developer Classic only

This activity is for use only in abstract processes. For details on creating an abstract process, see *Creating an Abstract Process*.

This activity stands in for an activity that would be used in an executable process.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes.*

To build an opaque activity:

1. Ensure that your BPEL process references a namespace for an abstract process. By default, a BPEL process is executable. See Tips for Working with Abstract Processes.

2. From the **Activity** palette, drag an opaque to the Process Editor canvas.

3. Add the activity to an appropriate container, if desired, or link another activity to it.

The following illustration shows an example of using an opaque activity.
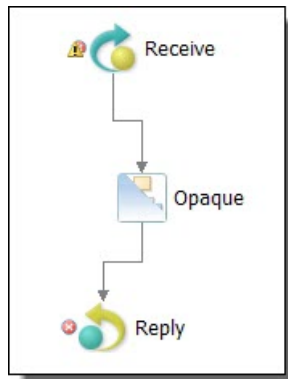
XML Syntax

```
<opaqueActivity standard-attributes>
    standard-elements
</opaqueActivity>
```

Example:

```
<opaqueActivity/>
```

Creating a Custom Activity

Select an activity or activity group previously saved in the Custom palette.

You can save any activity or set of activities from a BPEL file as a custom activity. You can shorten design time by reusing the custom activity in other BPEL files.

To create a custom activity:

1. Select one or more activities or containers on the Process Editor canvas.

2. Right-mouse click and select **Add to Palette**.
   The Custom palette appears, if it was previously empty. The selection is added to the Custom palette as *Custom1*.

3. Right-click the custom icon and select **Customize**.

4. Rename the custom icon by typing in the Name field.

5. You can use either Small (20 x 20 max.) or Large (32 x 32 max.) images for the item's icon.

6. Click **OK**.

Creating an Activity by Starting with a WSDL Interface

Select the activity type to create. Select Receive-Reply to create a matching Reply for the Receive. To activate the OnMessage activity, drop the Operation Wizard icon onto a Pick activity. For an OnEvent, drop the Wizard onto a process or scope Event handler. Select a fault name declared in the referenced WSDL file. Select an existing fault variable or create a new one. Select No Variable to use a message part for the activity.

Process Developer has a shortcut for you to create activities for an executable process. You can create a Web interaction activity by selecting an operation from the Participants or Interfaces view and completing a wizard.

For a key productivity starting point, see *Using the Participants View*.

Starting with a Partner Link Type Definition

If your WSDL file does not yet include partner link type definitions, you can select an operation from a port type.

1. Display the Partner Link Type element from a WSDL. The illustration shows an example of the loanProcess port type prior to the partner link type creation.
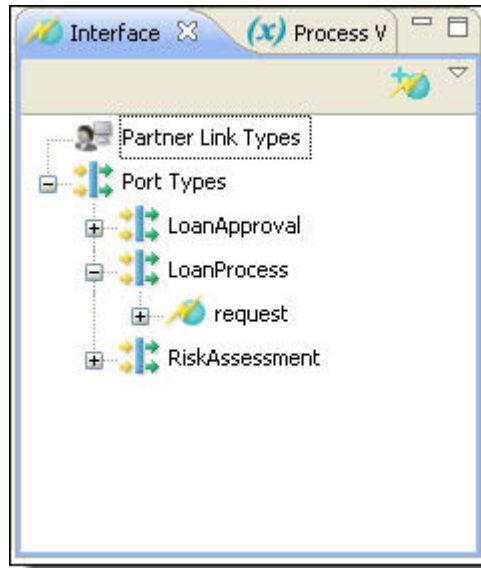
2. Select an operation to receive from, reply to, or invoke.

3. Drag the operation to one of the following locations:

   - Anywhere on Process Editor canvas to create a receive, reply, reply with fault, or invoke

   - Inside a scope on the Process Editor canvas if you want to create scope-level partner links or variables

   - Pick activity to create an onMessage and/or Reply

- Event handler to create an onEvent and/or Reply
  The list that appears on the Activity Type page varies depending on the location on which you dropped the operation. The following illustration shows the activities for a process or scope-level location, not for a pick activity or event handler. The Receive-Reply combination is listed because the WSDL operation has both input and output messages. Similarly, a Reply with Fault is listed because a fault is declared for the WSDL operation.
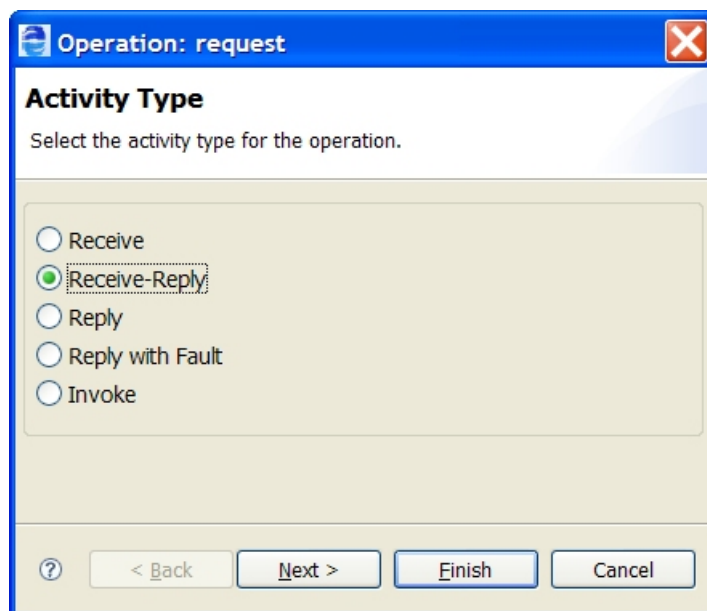
4. Select the activity type. As a shortcut, you can select Receive-Reply to set up matching activities at the same time. Click **Next**.
   If you are creating a Reply with Fault, see Creating a Reply with Fault.

5. Accept an existing process-level or ancestor scope partner link you have already defined in your process, if available, or type in a new one. A partner link identifies the roles played by the process and service. If you are creating a new partner link inside a scope, you can select Enclosing Scope for the declaration of the Location, making the partner link locally available only, instead of globally available to the process. Click **Next**.
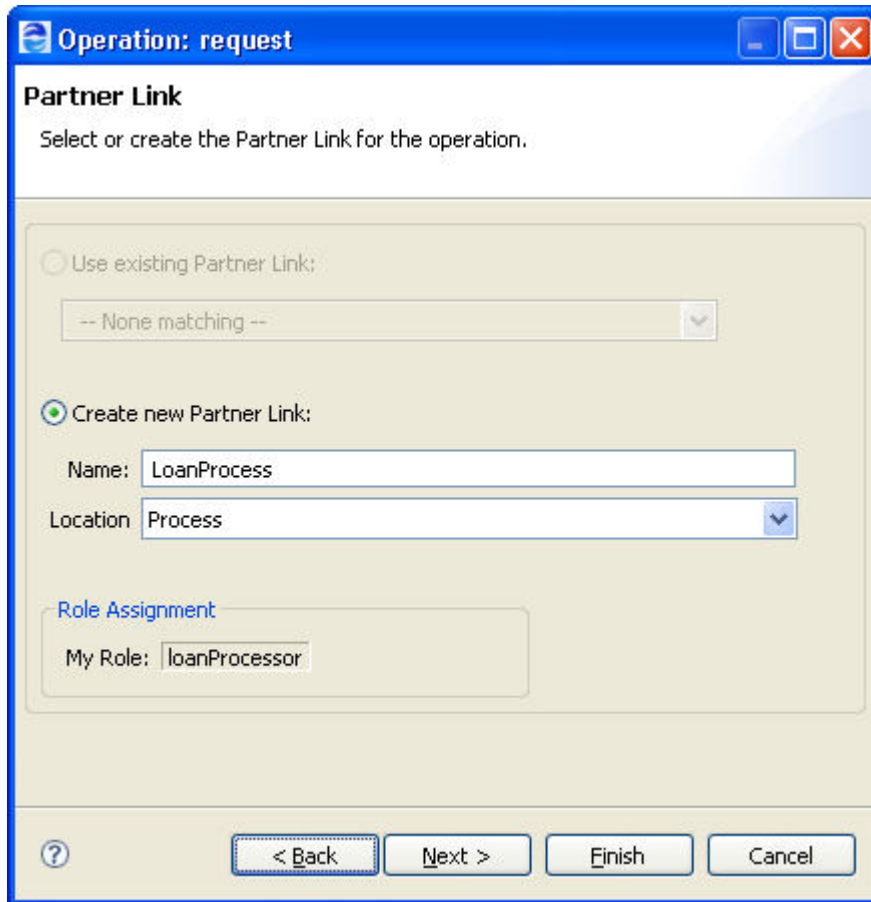
You can edit the partner link and other wizard information after you have finished the wizard.

6. Choose one of the following for the input variable, and then click **Next**:

- Use an existing variable name associated with the operation's input message

- Type in a new name for the input variable in your process. Select the Location for the declaration: Process or Enclosing Scope. Specify the variable type: Message or Element. If the variable is a message type with a single part defined by an element, you can select Element.

- Select No Variable if the variable is for an empty message or you want to create a fromPart or toPart for the activity. See *From Variable to Part* for details.
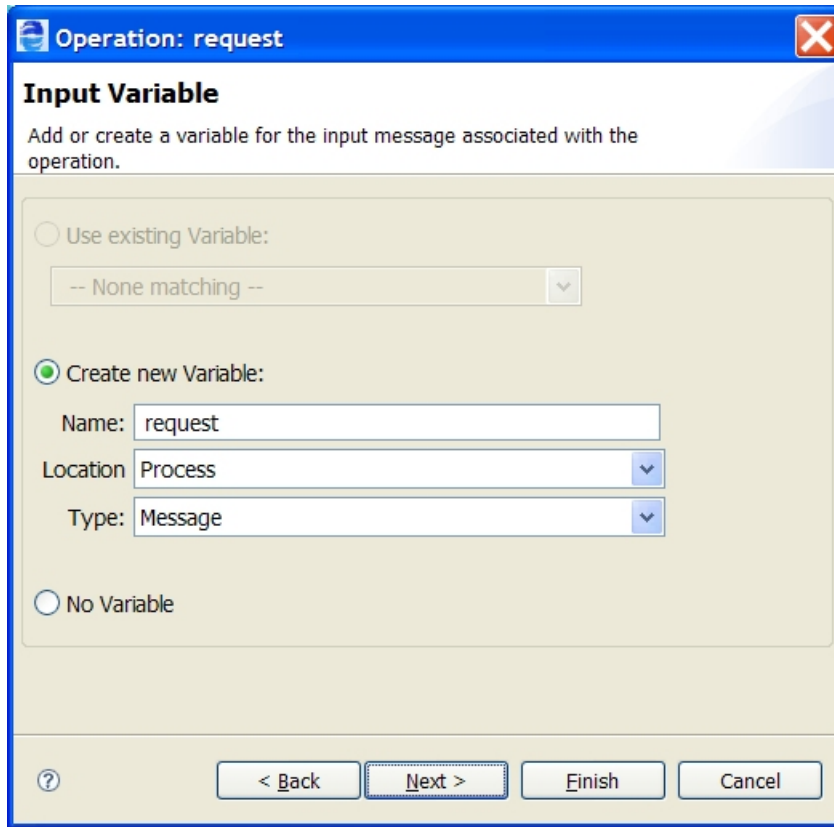
7. For an invoke or reply activity, repeat step 6 for the output variable name. Click **Finish**.

Creating a Reply with Fault

If you select Reply with Fault from the Operation Wizard, complete the following steps.

1. Select a fault declared for the operation, and click **Next**.
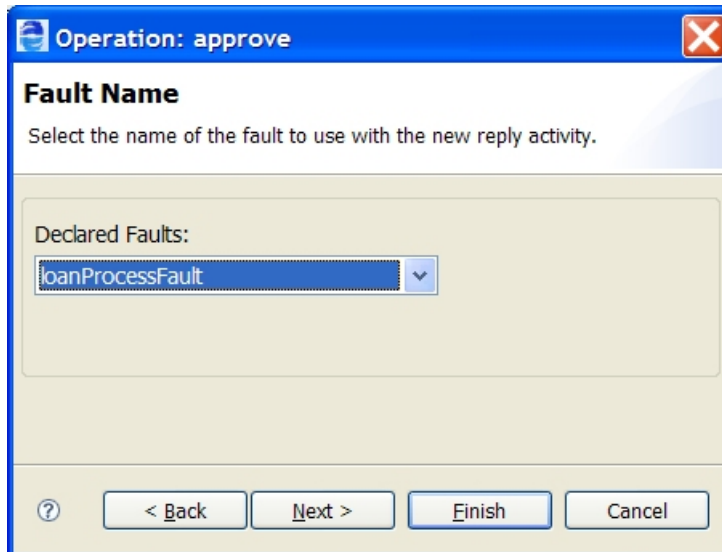
2. Make one of the following selections for the fault variable:
   • Use an existing variable name associated with the operation fault variable

- Type in a new name for the fault variable in your process. Select the Location for the declaration: Process or Enclosing Scope. Specify the variable type: Message or Element. If the variable is a message type with a single part defined by an element, you can select Element.
- Select No Variable if the variable is for an empty message or you want to create a fromPart or toPart for the activity. See *From Variable to Part* for details.

Starting without a Partner Link Type Definition

If the selected operation is not yet associated with a partner link type, you can select an operation from a port type. A wizard appears so you can create a partner link type definition and add it to an existing WSDL file or create a new WSDL file. For more information, see *Partner Link Type*.

C H A P T E R 8

Implementing a BPMN Gateway or Control Flow

Process Developer provides containers to structure activities. Each container has its own rules for how its child activities are executed.

The following illustration shows an example of a process using the several nested structures to give you a glance at how to use them.
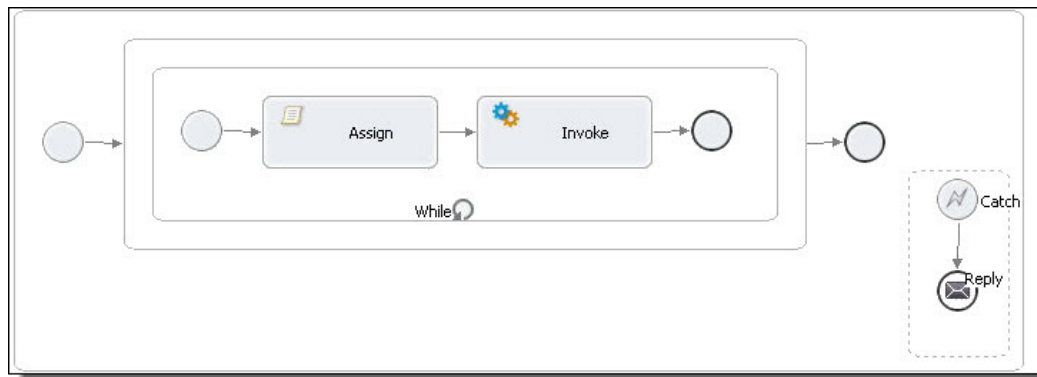
BPMN-to-BPEL Implementation of Gateways and Control Flow

BPMN-to-BPEL Implementation of Gateways and Control Flow

In many cases, a BPMN element can be implemented as one or more BPEL activities, described in the tables below.

The following tables are divided into BPMN types: gateways and control flow

For additional implementations, *BPMN-to-BPEL Implementation of Tasks and Events*.

Gateways

BPMN Gateway	BPEL Implementations
Exclusive	For details, see *Exclusive*.
Parallel	For details, see *Parallel*.
Inclusive	For details, see *Inclusive*
Complex	For details, see *Complex*.
Event-based	For details, see *Pick*

Control Flows

BPMN Control Flow	BPEL Implementations
Conditional Pattern	If
Fork Pattern	Two empty activities (parallel gateways) joined by links (equivalent of a BPEL flow activity)
Repeat Pattern	Repeat Until
Loop	While
Multi-Instance	For Each
Embedded subprocess	Scope

Different Ways of Structuring Activities

The BPEL language provides ways to structure and nest activities. Process Developer groups the structured activities in the Gateway and Control Flow palette of the Process Editor. Basic activities execute based on rules and conditions set by a parent container.

In addition to using structured containers, you can also structure activities by linking them with the edge tool. For more information, see *Using Links*.

Overview of Control Flow Items

The following table provides brief definitions of control flow activities.

Container Name	Description
Gateway	Allows incoming and outgoing links and various conditions for them, depending on the type. Equivalent to the BPEL empty activity.
Pick	Executes a selected activity based on an incoming message or a time-out event. Includes the Create Instance property to start a process.

Container Name	Description
Fork Join	BPMN equivalent of a BPEL flow activity, described above.
If	Executes a selected activity based on branch conditions
While	Executes an activity repeatedly until its condition evaluates to false
Repeat Until	Executes an activity at least once and repeats until its condition evaluates to true
Scope	Provides a context for set of activities, with correlation, compensation, fault handling, event handling, termination handling, and local partner links and variables
For Each	Iterates over a list of document elements
Sequence	(Process Developer Classic only). Executes activities in an ordered list
Flow	(Process Developer Classic only). Executes all activities in parallel

Ungrouping Selected Structured Activities

You can ungroup an If or Repeat Until activity, if they are not contained within a sequence. Right-mouse click on the activity and select **Ungroup**. Ungroup breaks the structured container and all of its contents into individual activities, links and gateways, shown in the same position. You can also select **Ungroup** for a sequence, which is the blue border enclosing activities.

Ungrouping an If or Repeat Until allows you to link to an activity that was contained. Without ungrouping, you can only link to the If or Repeat Until.

Gateway

The notation for a gateway control flow is BPMN only. This control flow is equivalent to a BPEL *Empty* activity.

A gateway represents a control point in a process. It plays two roles: join sequence flows together and split them apart. A gateway can play either role or both roles at the same time. The joins and splits are accomplished by incoming and outgoing links.

About Gateway Types

There are four types of gateways, discussed below. When you select a type, in the Properties view of a gateway, the join conditions of the incoming and outgoing links are automatically generated for that type. For a complex gateway, you can add the join condition manually. The default gateway type is Exclusive.

Based on the join conditions of the incoming or outgoing links, a gateway can be classified as one of the following:

- Inclusive (diamond with O)

The activity waits for at least one of the incoming links to evaluate to true and for the rest of the incoming links to be set with transition conditions. Any of the outbound links can have transition conditions, and any number of them can evaluate to true.

- Exclusive

The default type. The activity waits for at least one of the incoming links to evaluate to true and for the rest of the incoming links to be set with transition conditions. At most, one of the outbound links must evaluate to true. If, at run-time, more than one outbound link evaluates to true, the first link executes.

- Parallel (diamond with +)

The activity waits for all incoming links to evaluate to true. All outbound links are true (no transition condition is allowed out of a parallel gateway).

- Complex (diamond with asterisk)

The activity waits for all incoming links to be set (either true or false) and for the join condition specified within the complex gateway to evaluate to true. Any of the outbound links can have transition conditions, and any number of them can evaluate to true.

- Event-based. See *Pick*.

The BPEL structured activities, such as if, while, repeat until or for each easy-to-use alternatives to building decision-making with a gateway.

An exclusive gateway is a good choice as the source or target of a loop-back links. For details, see *Mutually Exclusive Transitions*.

Required Properties	Optional Properties
Gateway Type	Name.
	Join Condition.
	Suppress Join Failure.

Required Properties	Optional Properties
	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images
	Execution State.
	Extension Attributes and Extension Elements.
	Mutually Exclusive Transitions

Building a Gateway

Use the following procedure to build a gateway:

1. From the **Gateway** palette, drag a gateway to the Process Editor canvas.

2. Drag an activity, such as invoke, either below or above the gateway to automatically link it into a sequence.

3. If desired, add another activity to the canvas and link the two activities.

4. For tips on designing with links, see *Adding a Link with a Transition Condition*.

5. As desired, add transitions conditions to links, and then select the corresponding Gateway Type from the Properties view.

Mutually Exclusive Transitions

"Mutually Exclusive Transitions" is a property of all activities, but we recommend that you do not set this property directly. The exclusive gateway automatically sets this property to Yes.

This property, when set to Yes, declares that outbound links are mutually exclusive. This allows you to create loop-back links and create links that break out of a structured activity. Since there is no way to validate that two or more link conditions will not cause simultaneous execution of a target activity, you must set this property. For details, see *Process Developer Extension for Links*.

The Mutually Exclusive Transitions property is set to No as a default on all activities except the exclusive gateway, where it is set to Yes.

Pick

BPMN Implementation: Event-based gateway

The Pick activity has at least one message or message part and can optionally contain one or more alarms. When the Pick executes, it blocks until one of its messages is received or until one of its alarms go off. Only a single message or alarm ever executes in a Pick since the receipt of the message or the firing of the alarm immediately causes all of the other branches to go dead.

For example, a service can implement the process of waiting for a response to a quote by using a Pick with an accept message, reject message, and a timeout. If the accept or reject message is not received within the timeout period, the alarm activity executes.

A Pick activity is a good choice for asynchronous activities.

You can also use a Pick activity to start a process with a set of messages and no alarm. This special form of Pick sets the Create Instance property to Yes and executes when the first message arrives.

Required Properties	Optional Properties
For onMessage: Partner Link Operation Variable or From Part See *From Part to Variable*	Name. See *Selecting Activity Label* Create Instance Correlations. *Correlation* Join Condition. See *Creating a Join Condition for an Incoming Link* Suppress Join Failure. See *Process Properties* Message Exchange. See *Message Exchange Declaration*.
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
For onAlarm: Alarm Expression Alarm Type	*Setting Visual Properties and Using Your Own Library of Images*
	Execution State. See *Viewing the Execution State of an Activity or Link*.

To build an onAlarm branch of a Pick

1. From the **Gateway** palette, drag an **Event-based gateway** activity to the Process Editor canvas.

2. Select the onAlarm branch.

3. In the Properties view, select an Alarm Type: Duration or Deadline.

4. Click the `Dialog (...) Button` next to Alarm Expression.

5. In the Expression box, specify one of the following:

   a. For a deadline, enter an `xsd:datetime` expression, such as `'2010-12-12T12:00'`

   b. For a duration, enter an `xsd:duration` expression, such as `'PT1S'`

   c. For details and examples, see *Deadline and Duration Expressions*.

6. Drag an activity, such as a start, end, or none, to the On Alarm icon, and optionally select activity properties to reflect the actions to take when the alarm goes off.

To build an onMessage branch of a Pick

1. From the **Gateway** palette, drag an **Event-based gateway** activity to the Process Editor canvas.

2. Select the OnMessage branch.

3. In the Properties view of the OnMessage, create a new participant, if needed, or select one from the list.

4. Drag an activity, such as a Script Task (an assign), to the On Message branch, and select activity properties to reflect the actions to take when the message arrives.

The following illustration shows an example of a Pick activity.
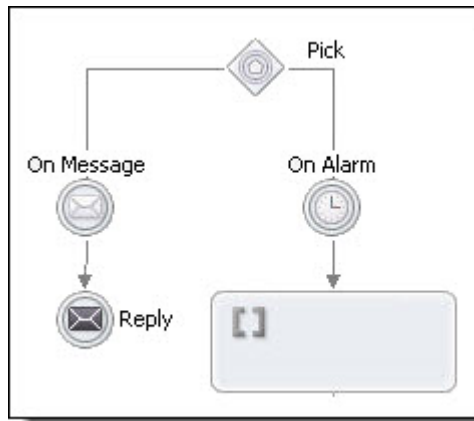
XML Syntax

```
<pick createInstance="yes|no"? standard-attributes>
    standard-elements
    <onMessage partnerLink="NCName"
        portType="QName"? operation="NCName"
        variable="BPELVariableName"?>+
        messageExchange="NCName"?>+
     <correlations>?
        <correlation set="NCName" initiate="yes|join|no"?>+
     </correlations>
     <fromParts>?
        <fromPart part="NCName" toVariable="BPELVariableName" />+
     </fromParts>
     activity
    </onMessage>
    <onAlarm>*
     (
        <for expressionLanguage="anyURI"?>duration-expr</for>
          |
      <until expressionLanguage="anyURI"?>deadline-expr>
      </until>
      )
      activity
    </onAlarm>
</pick>
```

Example:

```
<pick name="pick1" createInstance="yes">
  <onAlarm> <for>'PT30S'</for>
  </empty>
  </onAlarm>
  <onMessage partnerLink="customer"
    portType="lns:loanServicePT" operation="request"
    variable="request" >
    <reply activity>
  </onMessage>
</pick>
```

Fork Join

The notation for a fork join container is BPMN only. This control flow is equivalent to a BPEL Flow activity.

When you drag a fork join to the canvas, it contains two paths by default. You can add additional paths as needed. Each path contains one or more activities. Each path executes in parallel.

Required Properties	Optional Properties
none	Name.
	Join Condition.
	Suppress Join Failure.
	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images
	Execution State.
	Extension Attributes and Extension Elements.

To build a Fork Join:

1. From the **Control Flow** palette, drag a **fork join** to the Process Editor canvas.
2. Drag an activity, such as Receive task, to one path.
3. Continue dragging the other activities to the same or other path.
4. Add a new path by right mouse clicking the Fork Join and selecting **Add Path**.

The following illustration shows an example of a Fork Join activity containing two invokes executing in parallel.
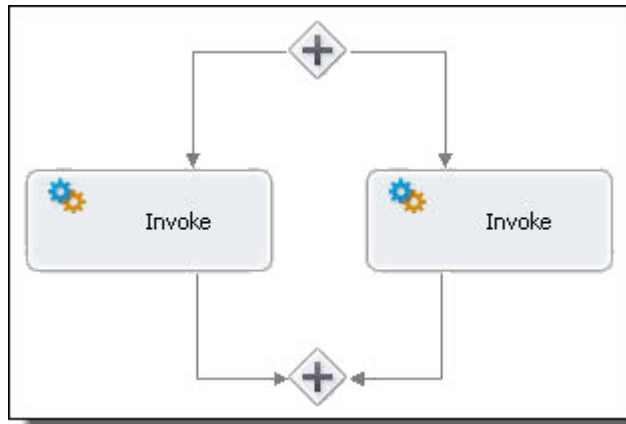
If

BPMN Implementation: Conditional Pattern

An If container executes an activity based on one or more conditions defined by the If and optional Else If elements, followed by an optional else element. The conditions are evaluated in order, and the first one to

evaluate to true has its activity executed. The if container is a good choice if all conditionals can be evaluated to a true or false condition.

If you do not define a branch, an implied Else executes an empty activity.

See also *Ungrouping Selected Structured Activities*.

Required Properties	Optional Properties
If Condition	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	See *Setting Visual Properties and Using Your Own Library of Images*.
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build an If:

1. From the **Control Flow** palette, drag a **Conditional Pattern** activity to the Process Editor canvas.

2. Double-click the **If Condition** container to open the Expression Builder.
   Alternately, in the Properties view, click the `Dialog (...) Button` next to If Expression.

3. Select the variables, functions and operators to build a Boolean expression. For details, see *Using the Expression Builder*.

4. From the **Activity** palette, drag an activity to the If Condition container, and select activity properties.

5. Repeat Steps 2 through 5 for the **Else If** or **Else** branches, if desired.

The following illustration shows an example of an If activity.
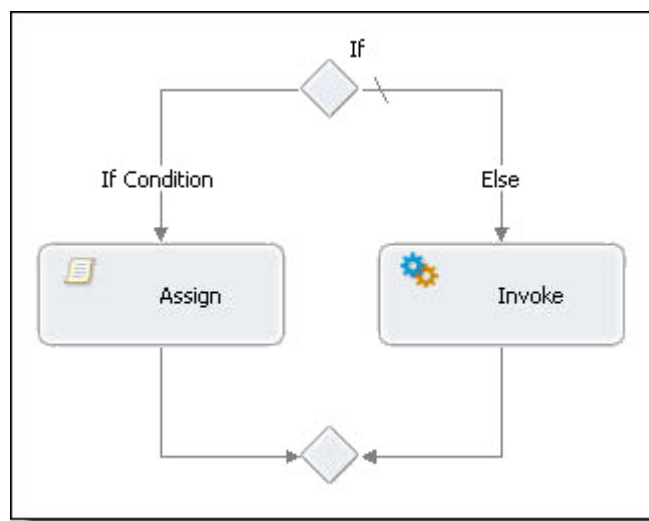
XML Syntax

```
<if standard-attributes>
    standard-elements
    <condition expressionlanguage="anyURI"?>
        bool-expr
    </condition>
        activity
  <elseif>*
    <condition expressionlanguage="anyURI"?>
        bool-expr
    </condition>
        activity
  </elseif>
  <else>?
   activity
  </else>
</if>
```

Example:

```
<if xmlns:inventory="http://supply-chain.org/inventory"
    xmlns:FLT="http://example.com/faults">
  <condition>
    bpel:getVariableProperty('stockResult',
     'inventory:level') > 100
  </condition>
    <flow>
      <!-- perform fulfillment work -->
    </flow>
  <elseif>
    <condition>
      <bpel:getVariableProperty('stockResult',
       'inventory:level') >= 0
    </condition>
     <throw faultName="FLT:OutOfStock"
        variable="RestockEstimate" />
  </elseif>
  <else>
     <throw faultName="FLT:ItemDiscontinued" />
  </else>
</if>
```

While

BPMN Implementation: Loop

The While activity executes an activity repeatedly until its condition evaluates to false.

In contrast to the Repeat Until activity, the While loop may not execute the contained activity at all.

Required Properties	Optional Properties
Condition	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*

Required Properties	Optional Properties
	See Setting Visual Properties and Using Your Own Library of Images.
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes.*

To build a While:

1. From the **Control Flow** palette, drag a **Loop** activity to the Process Editor canvas.

2. Double-click the Qhile to open the Condition Builder. For details, see *Using the Expression Builder.*

3. Drag an activity, such as a Scope or Invoke, inside the While.

4. Specify all the properties for each activity in the while.

XML Syntax

```
<while standard-attributes>
  <condition expressionLanguage="anyURI"?>
   bool-expr
  </condition>
    standard-elements
    activity
</while>
```

Example:

```
<while>
  <condition>
    $orderDetails > 100
  </condition>
  <scope>
    ...
  </scope>
</while>
```

Repeat Until

BPMN Implementation: Repeat Pattern

The Repeat Until activity executes an activity repeatedly until its condition evaluates to true. In contrast to the While activity, the Repeat Until loop executes the contained activity at least once.

See also *Ungrouping Selected Structured Activities.*

Required Properties	Optional Properties
Condition	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*

Required Properties	Optional Properties
	Documentation. See *Adding Documentation to a Process*
	See *Setting Visual Properties and Using Your Own Library of Images*.
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a Repeat Until:

1. From the **Control Flow** palette, drag a **Repeat Pattern** activity to the Process Editor canvas.

2. Double-click the Repeat Until to open the Condition Builder and build a Boolean expression. For details, see *Using the Expression Builder*.

3. Drag an activity, such as a Scope or Invoke, inside the Repeat Until.

4. Specify all the properties for each activity in the Repeat Until.

XML Syntax

```
<repeatUntil standard-attributes>
    standard-elements
    activity
  <condition expressionLanguage="anyURI"?>
   bool-expr
  </condition>
</repeatUntil>
```

Example:

```
<repeatUntil>
  <condition
   expressionLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:
     xpath1.0">$counter &gt; 0
  </condition>
    <sequence>
     <assign name="IncrementCounter">
      <copy>
       <from>$counter + 1</from>
       <to variable="counter"/>
      </copy>
     </assign>
     <wait name="WaitTwoSeconds">
      <for expressionLanguage=
        "urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
          'PT2S'
      </for>
     </wait>
    </sequence>
</repeatUntil>
```

Scope

BPMN Implementation: Embedded Subprocess

The Scope activity provides a context for a subset of activities. It can contain ault, event, and compensation handling for activities nested within it and can also have a set of defined variables and a correlation set.

A Scope can encompass a logical unit of work, making it manageable to execute, and then, if need be, reverse an activity. For example, if a customer cancels a paid travel reservation, the money must be returned, and the reservation must be canceled without affecting other reservations. By enclosing activities in a Scope, you can create the structure and conditions in which to manage activities as a unit.

Each Scope has a primary activity that can be a complex structured activity.

Required Properties	Optional Properties
Isolated. Refer to *Setting Isolated to "Yes" in a Scope*	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	See *Setting Visual Properties and Using Your Own Library of Images*
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a Scope:

1. From the **Control Flow** palette, drag an **Embedded Subprocess** activity to the Process Editor canvas. You can add a background color to the embedded subprocess if it is collapsed. Right-mouse click on the embedded subprocess and select Collapse Container. The color property will appear in the Properties view.

2. In the Properties view, set Isolated to Yes or No.

3. On the Process Editor canvas, right-mouse click the Scope activity to add declarations, including:

   - Correlation Set. For more information, see *Correlation*.
   - Variable. For more information, see *Using Variables*.
   - Participants. For more information, see *Using the Participants View*.

4. From the **Handlers** palette, add event fault and compensation handlers:

   - Compensation Handler. For more information, see *Compensate* and *Compensation*.
   - Event Handlers. For more information, see *Event Handling*.
   - Fault Handlers. For more information, see *Fault Handling*.
   - Termination Handler. For more information, see *Using a Termination Handler for a Scope*

5. From the **Task** palette, drag activities to the Scope container, and select activity properties.

The following illustration shows a sample scope activity with all handlers displayed.
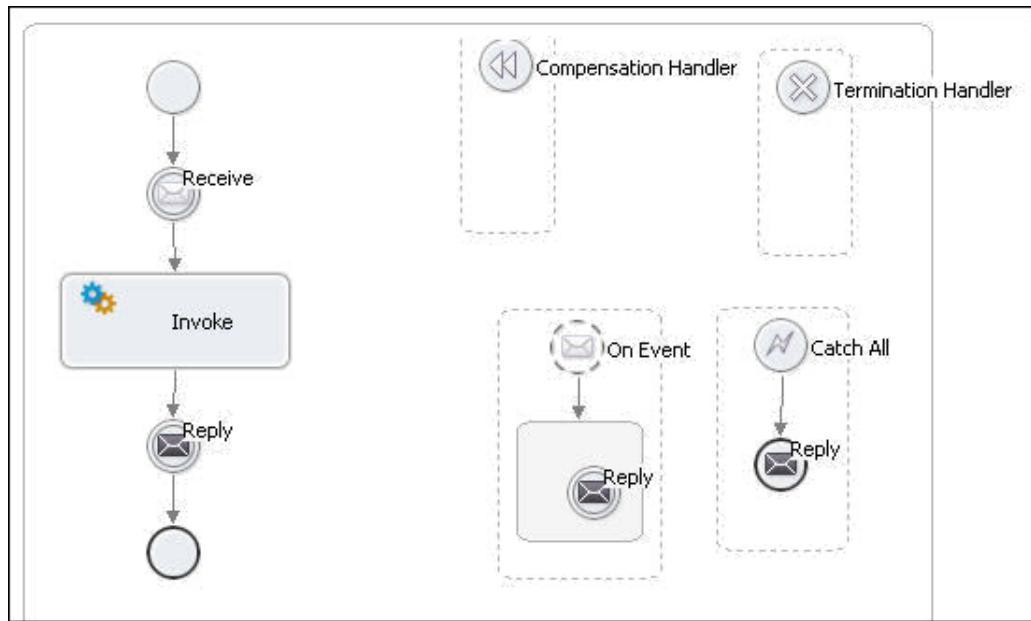
You can save an activity, or group of activities to the Custom palette for reuse. Because a Scope can be complex, it is a good candidate for reuse. For more information, see *Creating a Custom Activity*.

XML Syntax

```
<scope isolated="yes|no"? exitOnStandardFault="yes|no"?
  standard-attributes>
    standard-elements
    <variables>?
     ...
    </variables>
    <partnerLinks>?
     ...
    </partnerLinks>
    <correlationSets>?
     ...
    </correlationSets>
    <messageExchanges>?
     ...
    </messageExchanges>
    <faultHandlers>?
    ...
    </faultHandlers>
    <compensationHandler>?
    ...
    </compensationHandler>
    <eventHandlers>?
    ...
    </eventHandlers>
    <terminationHandler>?
     ...
    </terminationHandler>
  activity
</scope>
```

Example:

```
<scope isolated="no">
    <faultHandler>
     <reply>...</reply>
    </faultHandler>
    <flow>
      <invoke partnerLink="Seller" portType="Sell:Purchasing"
        operation="SyncPurchase"
```

```
              inputVariable="sendPO"
              outputVariable="getResponse"/>
          <invoke partnerLink="Shipper" portType="Ship:Orders"
              operation="OrderShipment"
              inputVariable="sendShipOrder"
              outputVariable="shipAck"/>
        </flow>
    </scope>
```

See also:

- *Setting Isolated to "Yes" in a Scope*
- *Lifecycle of a Scope*

Setting Isolated to Yes in a Scope

Two scopes can run at the same time. For example, at the end of the year, a bank executes two activities for each account: add interest and charge credit card costs. These activities can be executed in any order. However, since they both update the account, they cannot execute at the same time.

To ensure that variables referred to within each scope do not conflict, you can enable the isolated property. This property enables serialized access to variables, so that one scope has access to the shared variables and must finish using them before the concurrently running scope can use them.

Isolation is not enabled during compensation. For more information, see *Compensation*.

Using a Termination Handler for a Scope

BPMN Implementation: Cancel catch event

Activities can be terminated in a process through fault handling, early termination via a completion condition on a parallel `forEach` or through the execution of a Process Developer extension break activity. The Scope is the only activity that provides a means to control what happens when an activity is terminated. The termination handler for a scope is enabled while the scope is executing its main activity or its event handlers. The termination handler is disabled as soon as the scope completes its execution or if the scope's fault handlers execute. If a scope is eligible for termination, the termination of a scope begins by terminating its primary activity and all running event handler instances. Following this, the custom `<terminationHandler>` for the scope runs. If there is no termination handler for the scope, the default termination handler executes.

The default termination handler is:

```
<terminationHandler>
    <compensate/>
</terminationHandler>
```

For the main discussion of a scope's properties, see *Scope*.

Lifecycle of a Scope

When a Scope activity executes, it immediately registers any event messages or alarms with the engine prior to the execution of its child activity. Once registered, these messages and alarms are eligible to execute until the scope's child activity completes.

- If a fault occurs during execution of the child activity or one of the scope's events, the scope's fault handler attempts to catch the fault.
- If the fault cannot be caught by the scope, then it is rethrown. The scope is said to have faulted and is not eligible for compensation.

- If the child activity completes successfully, then the scope completes after uninstalling its event messages and timers, allowing them to complete if they are in the middle of executing.

A Scope is said to complete normally when it completes without a fault having been caught or rethrown. If a Scope completes normally, it is eligible for compensation.

For Each

BPMN Implementation: Multi-Instance control flow

The For Each activity contains a scope activity and executes it for a specified count. The execution iterations can occur in parallel or in sequence. The number of iterations to execute is determined by evaluating expressions for a start and final value. These values are inclusive, so a start of one and a final of 10 causes the enclosed scope to execute 10 times.

The expressions for the start and final values must evaluate to an `xs:unsignedint`. If it doesn't, a runtime exception of `bpel:forEachCounterError` occurs. If the final value is greater than the start value then the for each will not execute.

The counter name of the For Each is manifested as an implicit variable within the child scope. The initial value of the variable is set to the current value of the For Each's iteration, which ranges from the start value to the final value. This variable is accessible to all activities nested within the For Each's scope and appears as a variable within the child scope.

Using a Completion Condition for Processing N of M Branches

A For Each activity can include an optional *Completion Condition Expression* property that can in turn include an optional *Count Successful Branches Only* property.

The completion condition expression can be used to process at least N out of M branches. The expression evaluates to an `xs:unsignedint` value that is used to define condition of N out of M. It is evaluated once, when the for each begins its execution. If the `completionCondition` is not an `xs:unsignedint`, or it is determined that it cannot be met prior to executing, the For Each faults with a `bpel:invalidBranchCondition`. An example of this would be a `completionCondition` value of five when there are only three iterations of the for each.

The property *Count Successful Branches Only* can be set to Yes or No. If set to *Yes*, it means the Process Developer engine counts branches towards it completion condition that have completed normally (meaning the child scope of the for each iteration did not catch a fault and is eligible for compensation). The default value of No results in the For Each counting the completion of any child iteration towards its completion condition.

At the completion of each iteration, the For Each tests the number of completed scopes versus its completion condition (taking into account the Count Successful Branches Only setting as described above). If it there are not enough iterations remaining to fulfill its completion condition, the For Each faults with a `bpel:completionConditionFailure` and any remaining iterations do not execute, or are terminated in the case of the parallel for each.

A common use of the completion condition is to process only the minimum number of items necessary to complete a scope of work. For example, a BPEL process can call out to many partners in parallel, asking for approvals on an item. The process can require that only three approvals be needed to continue processing the item. Even though the process requested many approvals, it can break off the approval scope of work when only three approvals are received. It can then continue with further processing.

Parallel For Each

You can create a parallel or sequential For Each. In a parallel For Each, instances of the enclosed activity occur simultaneously. Each counter variable is initialized to one of the integer values in the count.

Use a parallel For Each for simultaneous processing of document parts. For example, you can process parts of a complex message variable, such as line items in a purchase order.

A default Message Exchange property that keeps together the correct pair of receive/replies or onMessage/replies, is automatically and implicitly declared in the scope. For details, see *Message Exchange Declaration*.

Sequential For Each

In a sequential For Each, instances of the enclosed activity execute one at a time.

Required Properties	Optional Properties
Counter Name	Name. See *Selecting Activity Labels*
Start Counter Value	Completion Condition Expression
Final Counter Value	Count Successful Branches Only
Parallel Execution Flag	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	See *Setting Visual Properties and Using Your Own Library of Images*
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a For Each

1. From the **Control Flow** palette, drag a **Multi-Instance** activity to the Process Editor canvas.

2. In the Properties view of the For Each, fill in the required properties:

    *   Parallel Execution Flag. Enable this setting to execute all instances of the activity concurrently. Be sure to read the Parallel For Each discussion above to learn the requirements for this type of execution. Disable this setting to execute the scope's activities sequentially.

    *   Add a Counter Name that identifies the enclosed scope's local variable. The default name is counter.

    *   Start Counter Value. Add an unsigned integer for the value. Type in a value or select the `Dialog (...) Button` at the end of the row to open the Expression Builder. For details, see *Using the Expression Builder*.

    *   Final Counter Value. See Start Counter Value above.

    *   If desired, add values for the optional property Completion Condition and enable Count Successful Branches to count only successful branches, as described above.

3. Fill in the properties for the Scope container.

4. Specify all the properties for each activity in the scope container. Be sure to set a message exchange property for matching receive/replies and `onMessage`/replies within the scope if parallel execution is enabled.

XML Syntax

```
<forEach counterName="NCName" parallel="yes|no">
   standard-attributes>
   standard-elements
      <startCounterValue expressionLanguage="anyURI">
      </startCounterValue>
      <finalCounterValue expressionLanguage="anyURI">
       </finalCounterValue>
       <completionCondition extension-attribute
        extension-element
        <branches expressionLanguage="URI"?
          countSuccessfulBranchesOnly="yes|no"?>
           an-integer-expression
        </branches>
       </completionCondition>
   activity
</forEach>
```

Example 1: 100 Executions, Executing if 10 Succeed

```
<forEach counterName="counter" parallel="no">
   <startCounterValue>1</startCounterValue>
    <finalCounterValue>100</finalCounterValue>
    <completionCondition>
      <branches countSuccessfulBranchesOnly="yes">
         10</branches>
    </completionCondition>
    <scope>...</scope>
</forEach>
```

Example 2: Serial For Each with Completion Condition
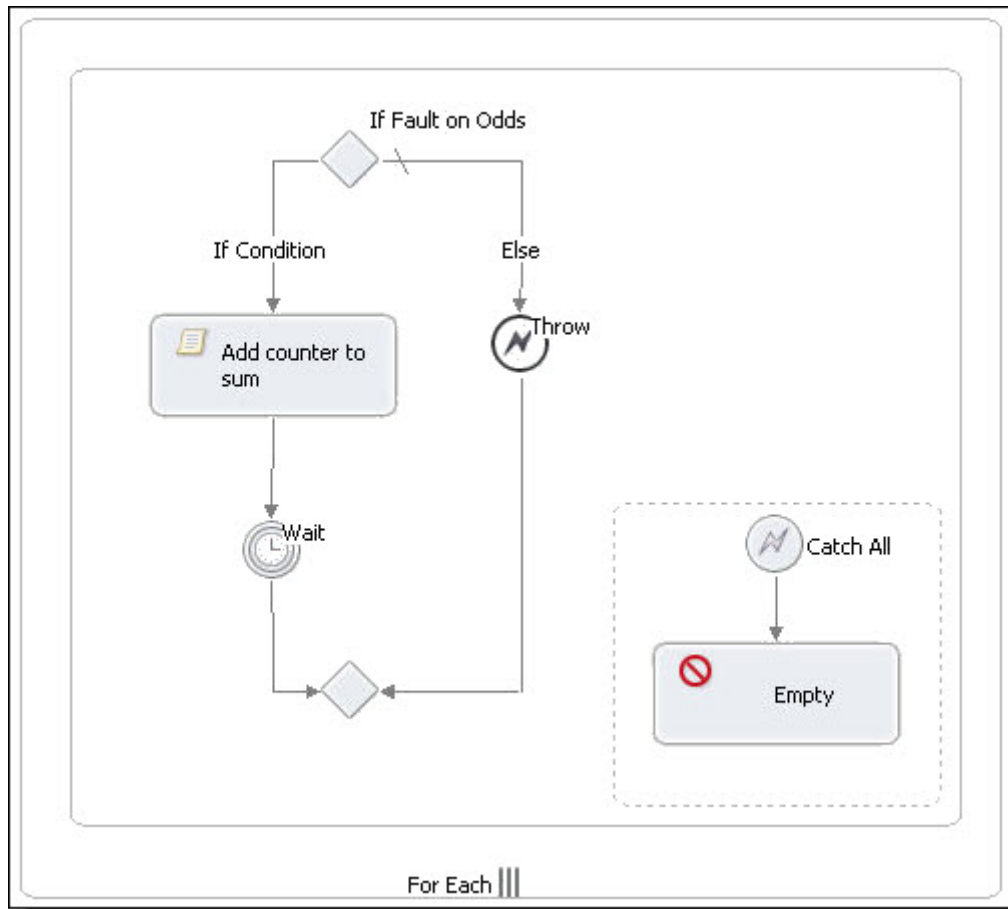
Example 3: Execution Thread of a Parallel For Each Activity

The following illustration shows the execution thread of a parallel For Each that has a counter value of 10. Notice that 10 instances of the enclosed scope are running in parallel.
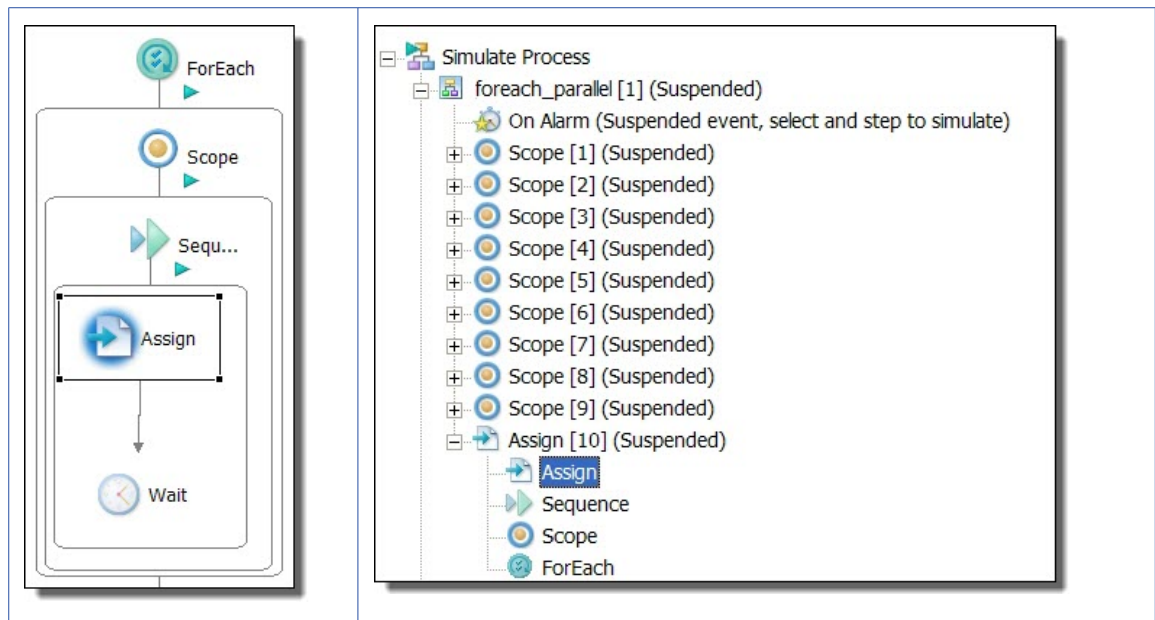
Faults Associated with a For Each Activity

The following faults can occur with a For Each activity:

- If a running process encounters an error with the counter variable, it throws a `bpel:forEachCounterError` fault.

- In a For Each activity, if a completion condition can never be true, the `bpel:completionConditionFailure` is thrown.

- If the integer value evaluated from the Count Completed Branches Only expression is larger than the Counter, the `bpel:invalidBranchCondition` fault is thrown.

Sequence

The Sequence container is displayed in Process Developer Classic layout style. If the BPMN style is in effect, there is no Sequence container in the palette. Every activity is contained within a hidden sequence to simplify designing on the canvas.

A Sequence container arranges and executes activities in an ordered list. This means that the first activity in a sequence executes, and when it is finished, the second activity begins.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*

Required Properties	Optional Properties
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	See *Setting Visual Properties and Using Your Own Library of Images*
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a Sequence:

Ensure that the layout style preference is set to Process Developer Classic.

1. From the **Container** palette, drag a **Sequence** activity to the Process Editor canvas.

2. Drag other activities into the sequence, organizing them from top to bottom or left to right.
   Tip: You can select activities on the Process Editor canvas and then right-mouse click to select **Create Container > Sequence**.

3. Set the properties for each activity in the sequence.

XML Syntax

```
<sequence standard-attributes>
    standard-elements
    activity+
</sequence>
```

Example;

```
<sequence name="SequenceExample">
 <receive name="receive1" partnerLink="customer"
    portType="lns:loanServicePT" operation="request"
    variable="request" createInstance="yes"/>
 <reply name="reply" partnerLink="customer"
    portType="lns:loanServicePT" operation="request"
    variable="approval"/>
</sequence>
```

Flow

The Flow container is displayed in Process Developer Classic layout style. If the BPMN style is in effect, there is no Flow container in the palette. The equivalent container is called Fork Join.

A Flow container executes all activities in parallel. This means you can define two or more activities, such as two Receive activities, to start at the same time. The activities start when the flow starts. The flow completes when all the activities in the container have completed.

For example, you can receive approval from both a seller and a buyer by adding the seller and buyer receive activities to a flow. Both receives must complete before the process executes another activity, such as invoking an order shipment service.

Flows can also contain links that allow you to introduce dependencies between activities to control the order in which they are executed. Refer to *Using Links* for more information.

The Flow is most useful for concurrent execution of activities.

Required Properties	Optional Properties
none	Name. See *Selecting Activity Labels*
	Join Condition. See *Creating a Join Condition for an Incoming Link*
	Suppress Join Failure. See *Process Properties*
	Comment. See *Adding Comments to a Process*
	Documentation. See *Adding Documentation to a Process*
	Setting Visual Properties and Using Your Own Library of Images
	Execution State. See *Viewing the Execution State of an Activity or Link*
	Extension Attributes and Extension Elements. See *Declaring Extension Elements and Attributes*.

To build a flow:

1. Ensure that the layout style preference is set to Process Developer Classic.
2. From the **Container** palette, drag a F**low** to the Process Editor canvas.
3. Drag an activity, such as Receive, inside the Flow.
4. Continue dragging the other activities into the Flow to execute concurrently.
5. Specify all the properties for each activity in the Flow.

The following illustration shows an example of a Flow activity.

XML Syntax

```
<flow atandard-attributes>
    standard-elements
    <links>?
      <link name="ncname">+
  </links>
  activity+
</flow>
```
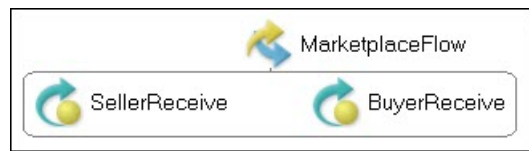
Example:

```
<flow name="MarketplaceFlow">
  <receive name="SellerReceive" partnerLink="seller"
      portType="tns:sellerPT" operation="submit"
      variable="sellerInfo" createInstance="yes">
  </receive>
  <receive name="BuyerReceive" partnerLink="buyer"
      portType="tns:buyerPT" operation="submit"
      variable="buyerInfo" createInstance="yes">
  </receive>
</flow>
```

CHAPTER 9

Using Variables

Process Developer provides several tools and shortcuts to define and use variables within a BPEL process, and to handle sample data.

Overview of Variables

A BPEL process receives, manipulates, and sends data through XML variables. Variables hold the messages exchanged between business partners and data used within the process.

For example, a process that receives a purchase order message from a buyer puts the message in an input variable. From there the variable content can be accessed by other operations. The following illustration shows an example.
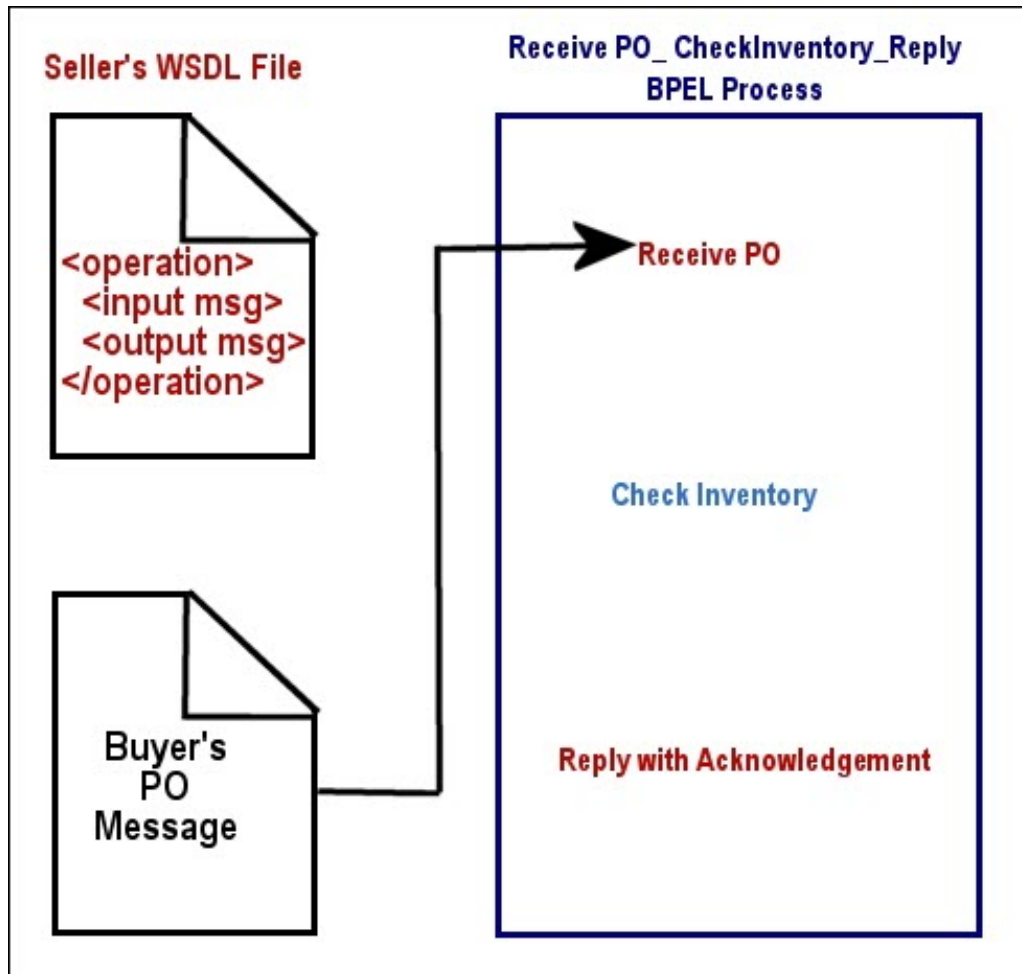
Variables are defined in one of the following ways:

- WSDL Message Types
- XML Schema Type
- XML Schema Element

Tips for using variables:

- You can initialize a variable when you declare it. For details, see *Initializing a Variable*.
- You can validate the value of variables against their definitions by adding a validate activity or a validate attribute in an assign activity. For details, see *Validate* and *Assign*.
- Variables can have attachments, and you can manipulate the attachments, as described in *Attachments*.

XML Syntax

```
<variable name="BPELVariableName"
    messageType="QName"? type=QName? element=QName?>+ from-spec?
</variable>
```

A variable name cannot contain the "." character. This character is reserved for use as a delimiter in WS-BPEL's default binding to XPath 1.0.

Adding a Variable

Define a new process variable based on a type.

You can add message type variables to your process automatically when you create a receive, pick, invoke, event handler, or reply activity from a WSDL. For details, see *Creating an Activity by Starting with a WSDL Interface*.

You can also add variables to your process manually. Before manually adding a variable, ensure that an appropriate WSDL namespace is defined. For more information, see *Importing WSDL, Schema, and Other Resources*.

You can declare global variables, accessible by the process as a whole, and you can declare local variables for use within a scope. A local variable can have the same name as a global variable, and if it has the same name, only the local variable is accessible within the scope.

To define a global variable:

1. In Outline view, right mouse click on Variables and select **Add > Declaration > Variable**.

2. Double-click the new variable, or in the Properties view, click **Dialog (…)** to open the **Definition** dialog.

3. In the dialog, select the variable type:

    - WSDL message

    - Schema type

    - Schema element

4. Select the name you need from the list that appears, and click **OK**.

To add an initial value to a variable, see *Initializing a Variable*.

To define a local variable:

1. Select a Scope from Outline view.

2. Complete Steps 2 - 4 above.

Variables are listed in the order they are created. To re-order the list, move a variable up or down in the Outline view.

WSDL Message Types

A WSDL message type variable uses a message type declared in a WSDL namespace that is referenced in your process.

The following illustration shows an example of a variable defined as a WSDL message type. The message type namespace is declared in the BPEL process.

```
BPEL Process
  <variable name = "BuyerPO"
  messageType = "lns:POMsg"/>


WSDL File
  <portType  name = "poPT">
    <operation name = "PO">
      <input message = "lns:POMsg"/>
  </operation>
</portType>
```

XML Schema Type

An XML schema type variable uses a simple or complex type of element built into the schema of the currently defined namespaces. For example:

```
<variable xmlns:props="http://example.com/shipProps"
        name="itemsShipped" type="props:itemCountType"/>
```

XML Schema Element

An XML schema element variable uses an element defined in the schema of the currently defined namespaces, as shown in the following example.

Adding Variable Properties and Property Aliases

Select properties defined in a WSDL to use in a correlation set.

Properties are useful on non-message variables as a way to isolate a process's logic from the details of a particular variable's definition. Using properties, a process can isolate its variable initialization logic in one place and then set and get properties on that variable in order to manipulate it. If the variable's definition is later changed, the rest of the process definition that manipulates that variable can remain unchanged.

A variable can have several properties, each defined as a *property alias*.

Properties

A property can be one of the following:

- Schema type like an xsd:int or an xsd:string
- Element

For example, a property might be a purchase order number or a customer Id. A property exists within a WSDL message transmitted during a Web service interaction or in any process variable. A single variable can contain multiple properties and a single property might exist in several variables.

Property Aliases

For each pairing of a variable and a property there exists a property alias that identifies how to extract the value of a property from the given variable, whether the variable part is a simple type, schema element, or complex type. If the part is an element or a complex type, an XPath (or other language) query is required to identify the location of the property within the element or type.

For details, see the following topics:

- *WSDL Syntax and Example for Property Names and Aliases*
- *Creating a Property Definition*
- *Creating a Property Alias*

Initializing a Variable

All variables must be initialized before use. A variable, or part of a variable, can be initialized in several ways, including receiving a message in an activity, assigning data in an assign or input mapping, or adding an initial value to the variable definition.

Variables in `receive`, `onMessage`, `onEvent`, and `inbound` invoke activities are automatically initialized. You can initialize other process variables when you declare them. Doing so allows you to skip an assign activity that would initialize a variable.

The initial value of a variable is validated against the schema or WSDL definition during process execution. You can also validate the variable by adding it to a validate activity. For details, see *Validating Variables*.

To add an initial value to a variable:

1. Add a new process variable, as described in *Adding a Variable*.
2. From the Outline or Process Variables View, select a variable.
3. In the Properties view, select the `Dialog (...)`) Button next to **Initial Value.**
4. In the **Variable Initialization** dialog, select the details for assigning a value to the variable. The selections are the same as those in the *From* side of a copy operation. For explanations of the *From Type* and related attributes, see *Assign*.

For a variable that is declared in a scope that contains a start activity, such as a Receive, you must not initialize the variable with a value that can change, such as a date-time function. This restriction allows for a pre-computed value for all process instances.

Viewing Variables

The Process Variables view has several features to help you easily define and use variables, such as:

- Quickly view variables in use by selecting an activity in Outline or Process Editor canvas. See *Quick View of Variables Used in Activities*.
- Search for all uses of a variable using *Find Where Used*. See *Finding Variables Where Used*.
- View a complete variable definition (message and parts, path, properties, elements and their attributes, and simple types). See *Understanding Icons, Symbols, and Descriptions of Variable Parts*.
- Drag and drop variables to create *Copy* operations for new or current assigns. See *Using the Process Variables View Options* and *Using Variables in a Copy Operation*.
- Add, load, generate, and edit sample data for message and element variables. See *Using Sample Data in Process Variables View*.

The following illustration shows an example of Process Variables list.
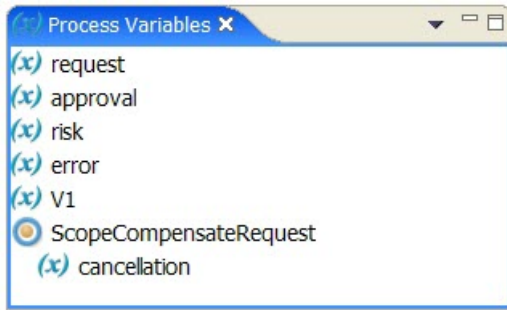
Quick View of Variables Used in Activities

When you select an activity on the Process Editor canvas or Outline view, the variables used in the activity appear selected in the Process Variables view.

Using the Process Variables View Options

Use the picklist of options on the Process Variables toolbar for convenience. The following table lists the options.

Option	Description
View Variable	Shows the variable list and lets you open a variable. Useful if the list is hidden. See *Opening a Variable to View its Definition*.
Show Variable List	Selected by default. If deselected, makes more room for open variables in the view by hiding the list of variables
Show Internal Variables	Deselected by default. If selected, displays the container variables associated with the implicit scopes created for data mapping within invokes, receives, and replies activities. Container variables are named `parameters` and `attachmentCopyResult`. For details, see *Input Variable*.
Show Variable Data	Displays variables in sample data view
Show Variable Definition	Displays variables in definition view
Layout Horizontal	Opens variables from left to right
Layout Vertical	Opens variables from top to bottom
Reload Variable Data	Displays last saved version of sample data

Tip: To rearrange the list of variables in Process Variables view, go to the Outline view and move variables up or down.

Opening a Variable to View its Definition

In the Process Variables view, you can open one or a selection of variables.

- To open one variable, double-click it
- To open a selection of variables, select them, and then select **Open All** from the right-mouse menu

- Select **Open All** to open all variables in the list

The following illustration shows an example of the `BuyerPO` variable opened.
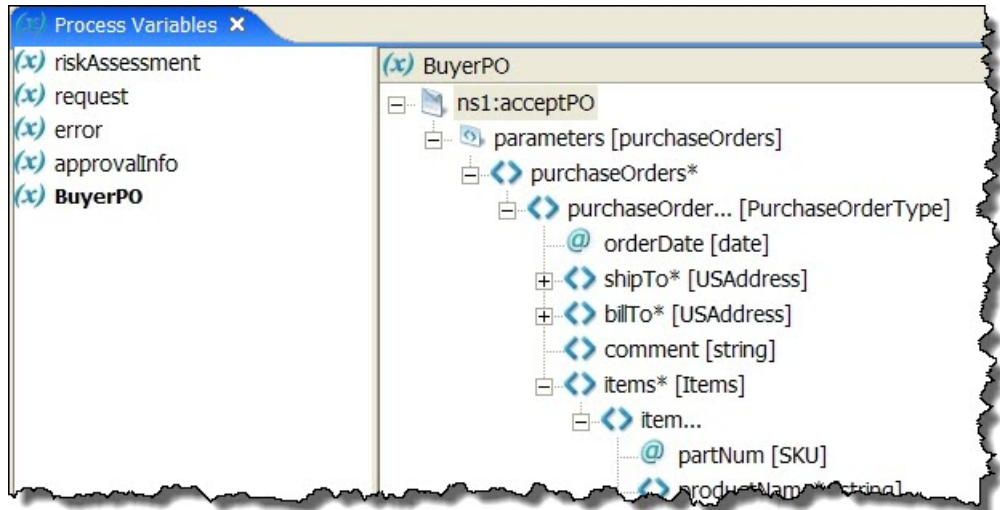
Viewing Variable Properties

Select a variable from Process Variables view or Outline to put the *Properties* view in focus.

The following table shows the required and optional properties for a variable.

Required Properties	Optional Properties
Name	Documentation.
Definition	Comment.
	Initial Value.
	Extension Attributes and Extension Elements.

Understanding Icons Symbols and Descriptions of Variable Parts

To understand each part of a variable, rest your mouse over it to view a description, as shown.
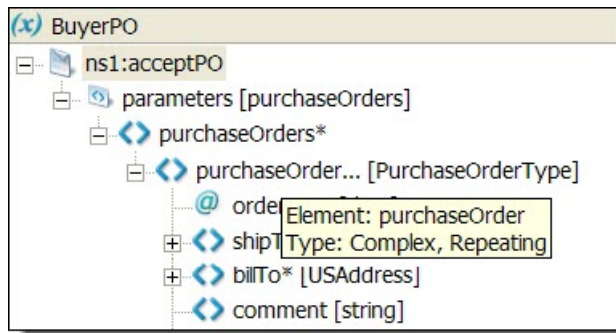
A variable definition can include:

- Message parts and properties
- WSDL schema types
- WSDL schema elements and attributes

An icon appears to the left of a variable part, property, element, or attribute. The icon, such as a pair of brackets (**< >**), is associated with a structural entity, such as an element.

Variables used in copy operations have additional visual information, as described in *Creating a Copy Operation Using Drag and Drop*.

In addition, to the right of a part, a symbol can appear, including:

- **Asterisk**, indicating a required field
- **Ellipsis**, indicating a repeating element

Deleting a Variable

You can delete a variable from your process.

1. Select **Process** from the Process Developer main menu.
2. Select **Delete > Variable**.
3. Select a variable from the Delete Variable dialog, and click **OK**.

Using Sample Data in Process Variables View

Select the source location for the sample you want to add. Select a workspace resource, such as sample data, WSDL, or XSL file.

The sample data values and files you add to WSDL messages are automatically loaded into Process Variables. For use during simulation, you can load sample data for message variables in the Process Variables view. A sample value serves to initialize an input, output, or fault variable for process simulation. These variables must be initialized prior to simulating execution of a process.

There are several ways to add sample values to variables:

- **Adding a Sample Data File to a WSDL Message**
  If you have added samples to WSDL messages in Participants or Interfaces view, the default sample value from the messages is automatically loaded into message type variables.

- **Editing a Single Sample Data Value for a Simple Type Message Part**
  If you have added sample data values to simple type message parts in WSDL messages, the default sample value from the messages is automatically loaded into message type variables. You can edit that value, or add a new value in Process Variables view.

- **Generating a Sample Data File**
  For complex variables, you can generate an XML document based on the root part or element of the variable.

- **Loading a Sample Data File in Process Variables View**
  For complex variables, you can select an XML document to load. You can edit data from the sample file and then save it or save it as a new file.

Each time you open a variable in the Data view, Process Developer displays the previously loaded data values.

Editing a Single Sample Data Value for a Simple Type Message Part

In the Process Variables view, you can add a sample data value for a simple type message part. When you are ready to test your process, the sample message part value is automatically loaded into your variable. You can also add additional values in Participants or Interfaces view as well as override the value during process simulation. For details, see *Adding or Editing a Sample Data Value for a Simple Type Message Part*.

To add a single sample data value:

1. Open a variable in the Process Variables list.

2. Right-mouse click on the variable name or part name, and select **Variable > View Data**.

3. Double-click the variable to open the **Edit Sample Data** dialog, as shown.
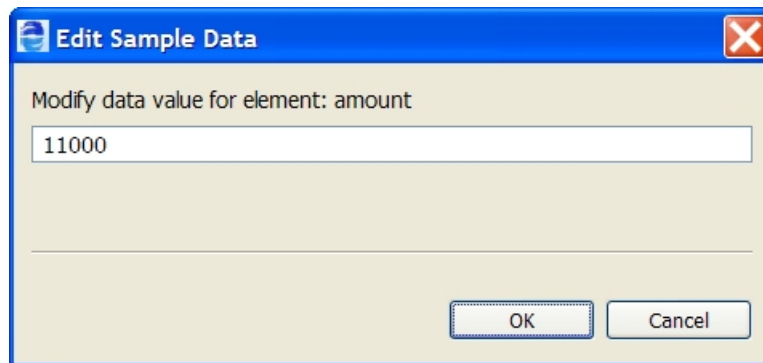
4. Type in a value and click **OK**. Notice that the value is displayed in the Data column of the open variable.

Loading a Sample Data File in Process Variables View

For convenience, you can prepare an XML data file for complex variables and then add the sample in Interfaces view. For details on sample data, see *Using Sample Data for WSDL Messages*.

To load a sample data file:

1. Create an XML file of sample data to represent a message or element structure from a WSDL namespace. The XML file must conform to the schema for the message type or element.

2. For convenience, add the sample data to your orchestration project. Samples can be used across BPEL processes. See *Generating a Sample Data File* and *Adding a Sample Data File to a WSDL Message* for more information.

3. Open a complex variable in the Process Variables list.

4. Right-mouse click on the variable name or part name, and select **View Data**.

5. Right-mouse click on a variable part and select **Load Data**.

6. Select one of the following:

- **Registered Samples Default**. Select this if you have added a sample data file to re-load the default file, if multiple data files exist, and you have changed the default.
- **From Project File**. Select this if you have sample XML files in a Project Explorer project.
- **From Registered Samples**. Select this if you have sample XML files for messages in Interfaces view.

7. After you select a file, you can expand the complex part to view data values.

Saving and Viewing Sample Data in Process Variables View

Once you have loaded or generated sample data for a complex process variable, you can right-click on the variable part and then save the data into a file or view it in a dialog.

On the variable part context menu, select **Save Data As** to create a new XML file.

Select **View XML Data** to open a dialog where you can easily view the variable contents.

Using the XML Data Wizard

For details on the Preferences page, see below.

Overview

The XML Data Wizard automatically generates the following types of data:

- **Sample data file** for a WSDL message that is a complex type
  Open the XML Data Wizard as described in *Generating a Sample Data File*

- **Literal contents** of a complex variable in the From side of a copy operation
  Open the XML Data Wizard as described in *Copy Operation Literal Contents Examples*.

- Literal contents of a complex variable when literal is selected as the type of *initial value* of a variable
  Open the XML Data Wizard as described in *Initializing a Variable*. The following dialog displays:
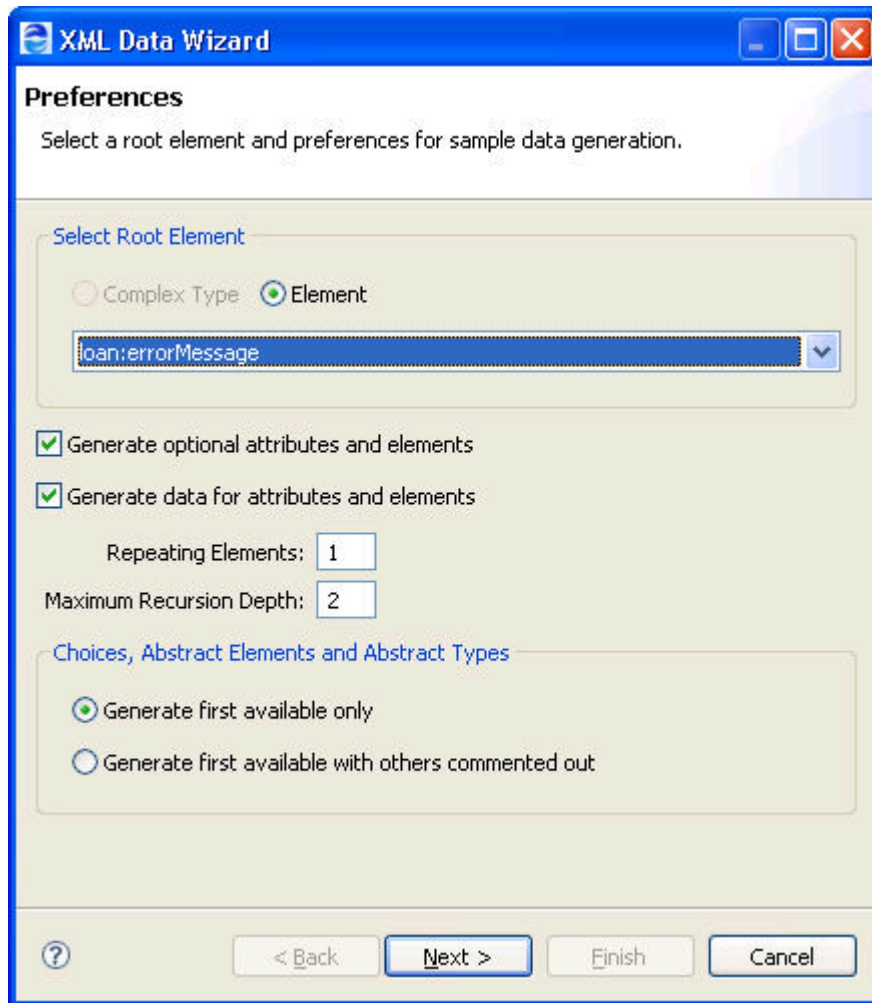
Make selections on the Preferences page as follows:

Selection	Description
Select Root Element	
Complex Type or Element	Complex type or element is pre-selected based on the variable selected. You can select any matching type or element from the picklist.
Generate optional attributes and elements	If the schema definition includes optional attributes (that is, `use="optional"`), and elements (that is, `minOccurs="0"`), you can include them in the data file, if desired. If you do not include them, you can experience a selection failure fault when you run the process, depending on the query used in selecting elements.

Selection	Description
Generate data for attributes and elements	Select this to generate appropriate values for elements and attributes. In some cases, such as derived simple types with regular expression patterns, the values are not schema valid. In most cases, the values provide a reasonable starting point for you to edit as necessary.
	If the element or attribute contains a fixed or default value it is automatically used. If a referenced type contains an enumerated restriction then the first value is selected from this restriction.
	Deselect this to generate empty elements and attributes. Generating elements without data does not produce a schema-valid sample. For variable initialization and copy operations, you can add your own values in the Literal Contents edit box after generating the data. For Interfaces and Process Variables sample data, you can open the generated XML file and edit it.
Repeating Elements	For variables with repeating elements, such as a purchase order with multiple items, you can specify how many items to generate for that element. The default is one.
Maximum Recursion Depth	Sets the recursion depth for a schema type that is recursive. The default depth is two, and the data generated is similar to the following:

	`<foo>`
	`    <child />`
	`    <foo>`
	`        <child />`
	`        <foo/>`
	`    </foo>`
	`</foo>`

Choices, Abstract Elements and Abstract Types	
Generate first available only	A Schema type can contain a choice construct that specifies that only one of the available choices can appear in the resulting document. The first available element is the first child element.
	A Schema element can contain one or more abstract elements. These elements cannot be created directly since they are abstract. Instead, the first available element is selected from its substitution group. The first available element is the first one encountered within any imported schemas.
Generate first available with others commented out	Select this to generate all available elements, with all but the first generated within `<!-- --->` tags.

Example One: Repeating Elements

Given the following schema definitions for element=ns:items:

```
<xs:element name="items" type="imp:ItemsType" />
   <xs:complexType name="ItemsType">
     <xs:sequence>
       <xs:element ref="imp:product" maxOccurs="unbounded" />
     </xs:sequence>
   </xs:complexType>
  <xs:element name="product" type="imp:ProductType" />
  <xs:complexType name="ProductType">
     <xs:sequence>
       <xs:element name="number" type="xs:integer" />
       <xs:element name="name" type="xs:string" />
     </xs:sequence>
   </xs:complexType>
```

The following literal contents (or sample data file) is generated when Repeating Elements is set to three:
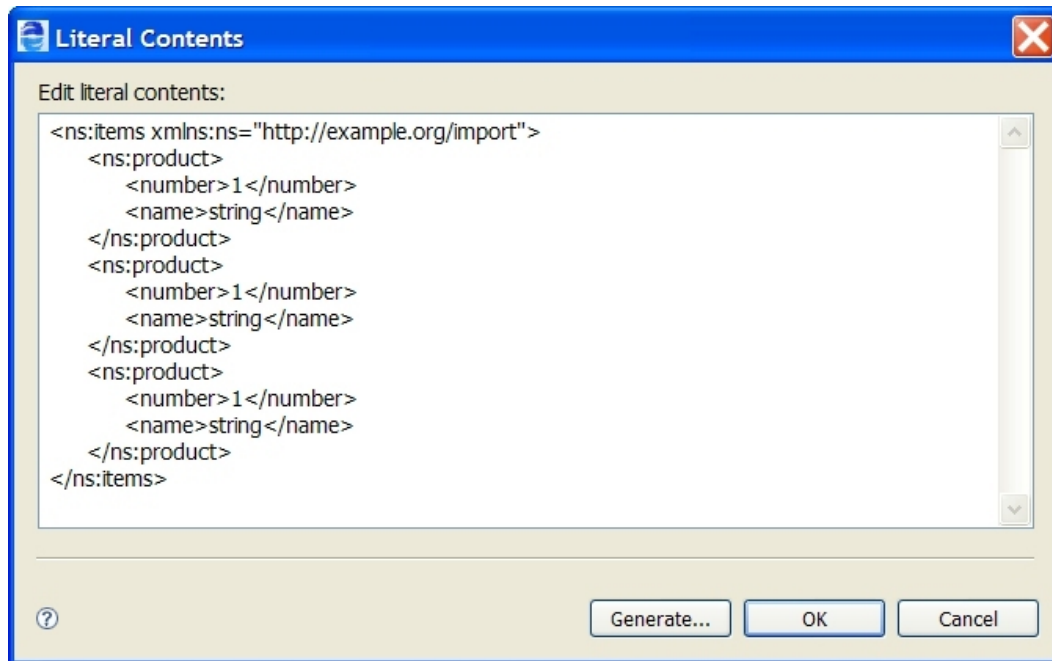
Example Two: Generate first available with others commented out

Given the following schema definitions for `element=ns6:productList`:

```
<xs:element name="productList">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="tns:product"
                 maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="product" type="tns:ProductType"
      abstract="true"/>
   <xs:element name="hat" substitutionGroup="tns:product"
      type="tns:ProductType"/>
   <xs:element name="glove" substitutionGroup="tns:product"
       type="tns:SubProductType"/>
   <xs:complexType name="ProductType">
      <xs:sequence>
        <xs:element name="color"    type="xs:string"/>
        <xs:element name="style"    type="xs:string"/>
        <xs:element name="price"    type="xs:decimal"/>
        <xs:element name="quantity" type="xs:positiveInteger"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="SubProductType">
      <xs:complexContent>
        <xs:extension base="tns:ProductType">
          <xs:sequence>
            <xs:element name="size" type="xs:int" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
   </xs:complexType>
```

The following literal contents (or sample data file) is generated :
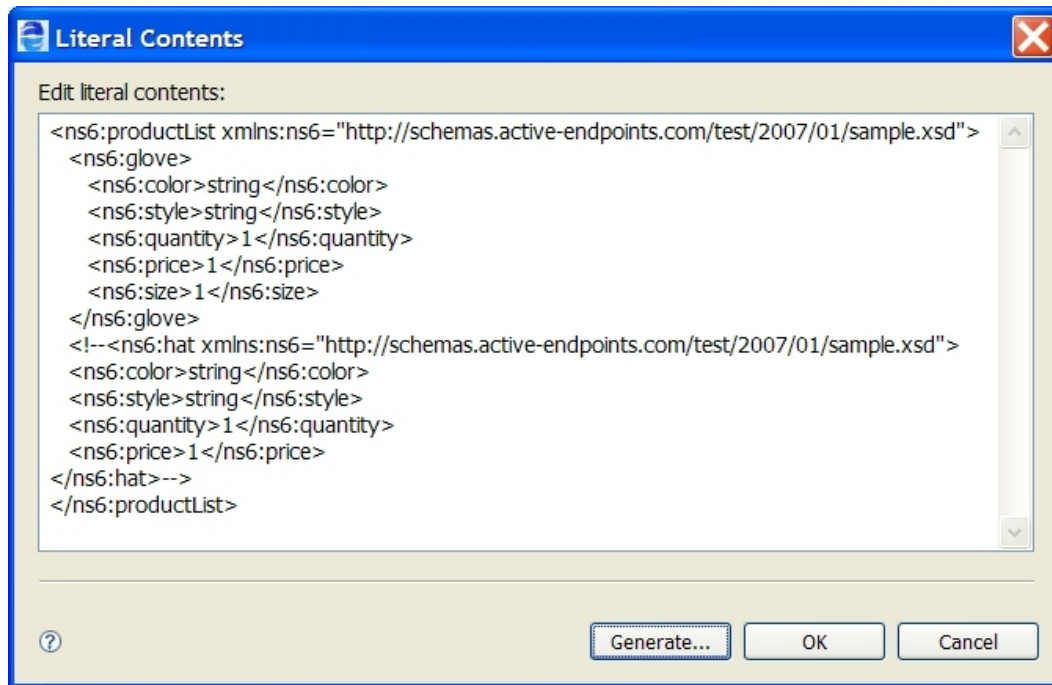
Finding Where Variables are Used

You can search for all occurrences of a variable in activities and links by using the **Find Where Used** option.

To search for occurrences of a variable from the **Process Variable** list, perform the following steps:

1. Open the project in Project Explorer.

2. From the menu bar, select **Window** > **Show View** > **Process Variables**.
   The **Process Variables** section appears with a list of process variables used in the project.

3. Select the variable that you want to find, right-click, and then select **Find Where Used** as shown in the following image:
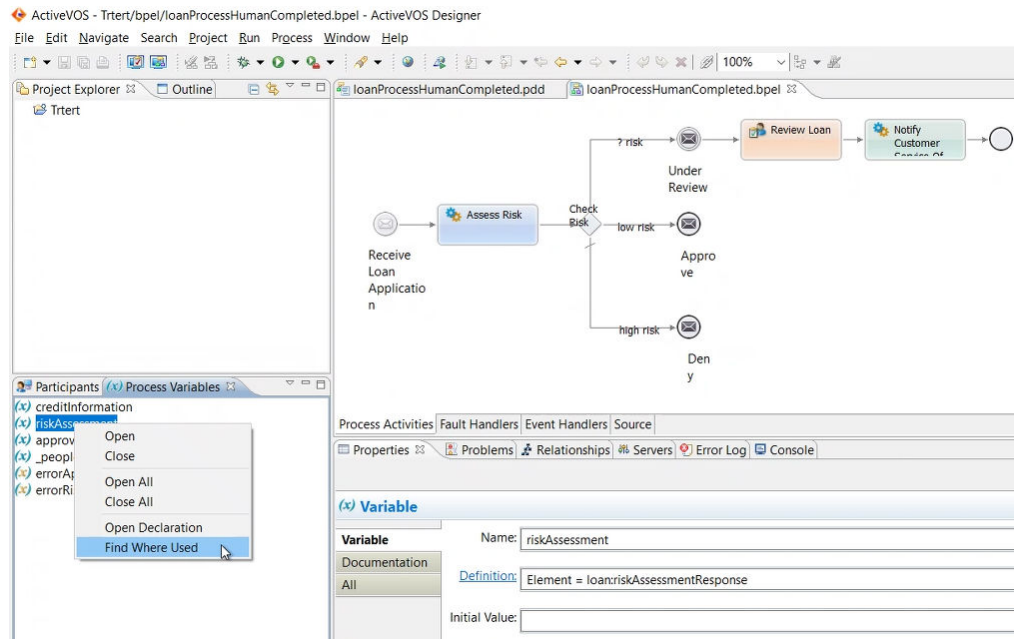
A list of activities and links where the variable is used appears on the **Search** tab.

You can also search for a variable that is used in Copy operations.

To search for occurrences of a variable in Copy operations, perform the following steps:

1. Open the project in Project Explorer.

2. From the menu bar, select **Window** > **Show View** > **Process Variables**.
    The **Process Variables** section appears with a list of process variables used in the project.

3. Select the variable name from the **Process Variables** section.
    The variable section appears with the variable parts and properties.

4. Right-click the variable and select **Copy** > **Find Where Used** as shown in the following image:
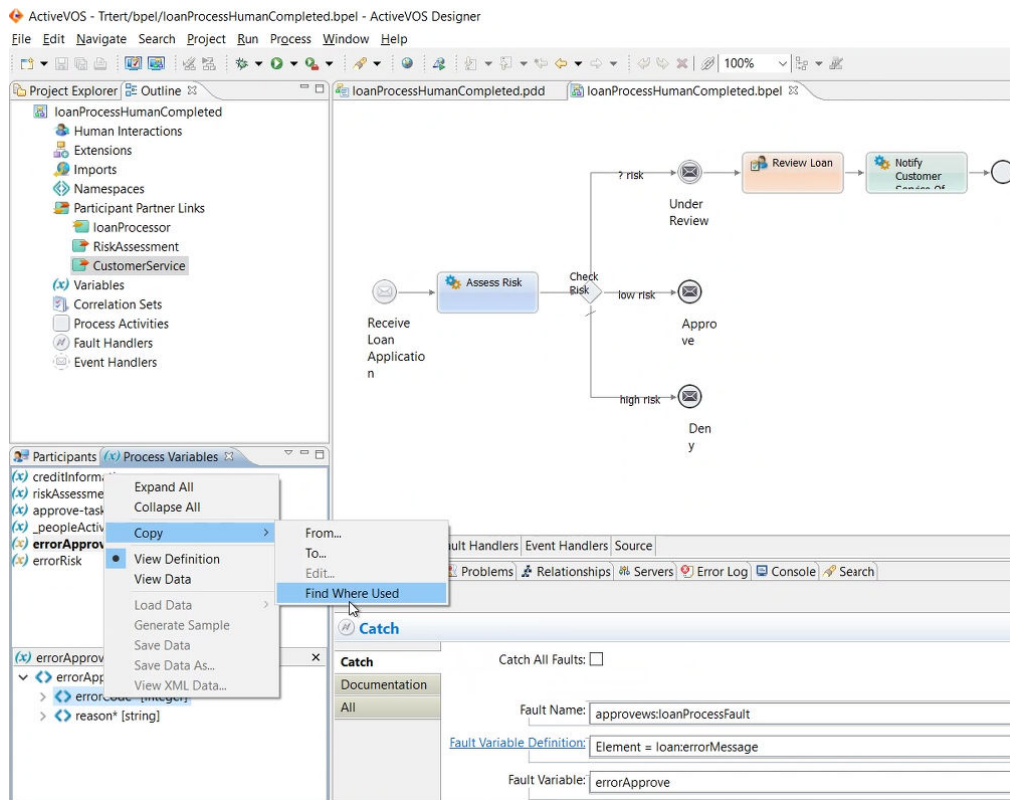
A list of occurrences of the variable in Copy operations appears on the **Search** tab.

Process Developer displays the search results and highlights the first result in the Process Editor canvas and Outline view. From the Outline view, you can select an activity, link, or Copy operation for editing.

Using Variables in a Copy Operation

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

Within the Process Variables view, you can create and edit Copy operations for a current Assign or create a new Assign automatically.

You can create a Copy operation by using a variable's context menu or by dragging and dropping a variable part to another variable part.

Select a FROM Type and then complete the Copy operation by making other appropriate selections. Add, edit, copy, delete, and reorganize Copy operations and scripts for the selected Assign activity.

For more information, see:

- *Creating a Copy Operation Using a Context Menu*

- *Creating a Copy Operation Using Drag and Drop*

- *Selecting a Copy Operation to Edit*

Creating a Copy Operation Using a Context Menu

Tip: Use this procedure to create any type of Copy operation. For a quicker way to create a variable-to-variable Copy operation, see *Creating a Copy Operation Using Drag and Drop*.

1. Do one of the following:

   • Select an assign activity from the Process Editor canvas, if you want to add a Copy operation to it. The Copy is added after any current Copy operations.

   • Go to Step 2 if you want to create a new assign activity automatically.

2. In the Process Variables list, open a variable for either the *From* or *To* clause of the Copy operation.

3. Right-mouse click and select **Copy > From** to create the *From* clause or **Copy > To** to create the *To* clause.

4. In the **Copy Operations** dialog, complete the remaining clause, and click **OK**.

In the Process Editor canvas and Outline view, the current or new assign activity is selected.

Creating a Copy Operation Using Drag and Drop

You can create the following types of Copy operations by dragging and dropping:

```
From Variable [Part] To Variable [Part]
From Variable Property To Variable Property
```

For details, see *Assign*.

To create a Copy operation by drag and drop:

1. Do one of the following:

   • Select an assign activity from the Process Editor canvas, if you want to add a Copy operation to it. The Copy is added after any current Copy operations.

   • Go to Step 2 if you want to create a new assign activity automatically in the currently selected Scope.

2. In the Process Variables list, open the two variables for the *From* and *To* clauses of the Copy operation.

3. Drag the variable, part, or property of the *From* variable to the appropriate variable, part, or property of the *To* variable.
   Tip: Before you release the mouse button, you can view the Copy specification in the Process Developer status bar.

After you release the mouse button, Process Developer shows the results by highlighting the parts in use. Also, the *From* part icon has an outward arrow and the *To* part icon has an inward arrow, as shown.
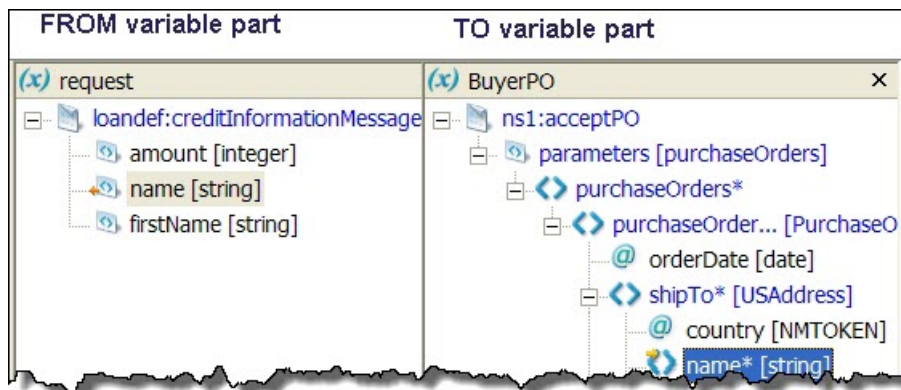
In the Process Editor canvas and Outline view, the current or new Assign activity is selected.

Selecting a Copy Operation to Edit

You can edit a Copy operation by selecting **Copy > Edit** from a variable's right mouse menu.

If a variable has multiple Copy operations, the **Copy Operations** dialog opens so that you can select an operation to edit.

Using Variables Based on WSDL Fault Messages

Fault names and variables based on WSDL fault messages are used within BPEL fault handling constructs. For details on defining and using fault names and variables within a scope or within the process as a whole, see *Fault Handling*.

Fault names and variables based on WSDL fault messages are used within BPEL fault handling constructs.

Mapping WSDL Message Parts in Web Service Interaction Activities

In a Web service interaction activity, such as a receive, onMessage, onEvent, invoke, or reply activity, you usually select a WSDL message type variable for the input or output variable. However, if the WSDL operation uses a message containing exactly one part that itself is defined using an element, the Web service interaction activity can use the `<fromPart>` element for an input variable and a `<toPart>` element for the output variable. The *fromPart* or *toPart* declarations are used in place of the activity's input or output variable declaration.

Validating Variables

Process Developer provides several explicit ways to validate the value of variables during simulation or runtime. These are as follows:

- **Validate activity**
  The validate activity contains a list of process variables and validates the value of each one against its associated XML or WSDL data definition. For details, see *Validate*.

- **Validate attribute in an assign activity**
  This attribute works the same way as the validate activity. When validate is enabled in an assign activity, Process Developer validates all variables used in all of the assign's copy operations.

- **Keep Source Element Name attribute in a copy operation** for validating the value of an element-based variable in the To side of a copy operation .
  The *Keep Source Element Name* attribute works specifically on an element-to element copy operation, and is not applicable for any other type of copy operation. Process Developer validates that the root elements are valid against their XML schema or WSDL definition, without performing validation on an entire variable, since the destination element may be only one of many elements in the variable. This type of copy operation requires that the source and destination elements be part of the same schema

substitution group. A substitution group allows elements to be substituted for other elements. For an example, see *Copy Operation Query and Expression Examples*.

- **Preference setting for validating input/output message variables**
For details, see *Simulation Preferences*.

If a variable value is invalid, the `bpel:invalidVariables` fault is thrown.

Working with Variable Attachments

Variables can have unstructured data that come into the process as attachments. For details in adding, removing, simulating and debugging with variable attachments, see *Attachments*.

CHAPTER 10

Attachments

Many messages used in business processes contain references to unstructured data. For example, an insurance business process executing a claim request can reference a picture of a damaged vehicle. To include such unstructured data in Process Developer, you can attach a picture, document, spreadsheet, or other file type to a variable in a BPEL process.

You can receive, send, and copy attachments from one variable to another. You can also copy the contents of an attachment or attachment property to a process variable by using custom functions.

Adding an Attachment

You can add attachments to any process variable, as well as manipulate attachments with a set of functions, described in *Custom Functions for Manipulating Attachments*.

Currently, to view attachments in Process Developer, you can add them as a simulation property, described in *Adding an Attachment for Simulation*.

Be sure to initialize a process variable, if it has not been initialized, before copying attachments to it. See *Initializing a Variable* for details.

Adding an Attachment for Simulation

You can add an attachment to a variable to simulate the attachment that a running process would normally receive and send. You can also add properties to the attachment and then manipulate properties with the `getAttachmentProperty` function. For details, see *Custom Functions for Manipulating Attachments*.

For a discussion of simulation, see *Simulating Execution of a BPEL Process*.

To add attachments to a variable for simulation:

1. From the Process Editor canvas, select a receive, invoke, onMessage, or onEvent activity.
2. In the Properties view, under Simulation, do one of the following:
   - For a receive, onMessage, onEvent, click the `Dialog (...) Button` at the end of the *Input Attachments* row
   - For an Invoke, click the Dialog (...) Button at the end of the *Output Attachments* or *Fault Attachments* row
3. In the **Input/Output/Fault Attachments** dialog, select **Add**.

4. In the **Attachment** dialog, select either *File System* or *Project* for an attachment location, and browse to select an attachment.

5. Notice that known content properties are identified. For example, if you attach a GIF image, the Content Type is *image/gif*.
Note that the *X-Size* property that is added to indicate the size of the attachment in bytes.

6. If desired, edit the known properties, or add a new property as follows:

   a. Select **Add** to open the **Attachment Properties** dialog.

   b. Type in a *Property Name* and *Value*. Note that the property must be a known property of the attachment.

   c. Click **OK**.

7. Click **OK** to close the **Attachments** dialog.

8. To add another attachment, select **Add** on the **Input/Output/Fault** dialog, and continue adding an attachment.

After adding attachments, start simulation, as described in *Starting and Ending Simulation of a BPEL Process.*

During simulation, open the process variables that have attachments. As the process receives, sends, or copies attachments, you can view the results in the Process Variables view.

Attachment properties must conform to known properties. For details, see abx:getAttachmentProperty in *Custom Functions for Manipulating Attachments*.

Remote Debugging with Variable Attachments

You can deploy a BPR file to the Process Server and debug it. For details on remote debugging, see *Debugging Remote Processes Running on the Server*.

To send a message to the process that includes attachments, you need a SOAP tool that has this capability. Refer to the Informatica support forums and Samples page for a discussion of SOAP tools.

You can also add, remove, and download attachments from variables during remote debugging.

To add an attachment during remote debugging:

1. Start a remote debug session.

2. In the Process Variables view, open a variable in Data view.

3. Right-click on the variable or variable part, and select **Add Attachments**.

4. In the **Attachment dialog**, select one or more files.

The variable with attachments looks like the following example.
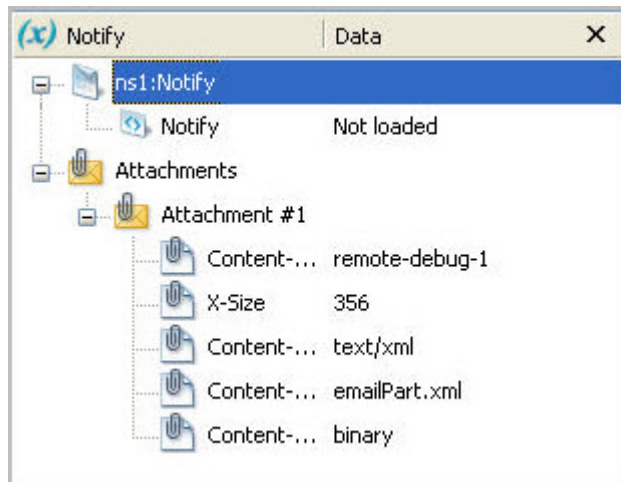
In the Process Console of your Process Developer engine, you can open the Active Process Detail View to view the new attachment.

To remove an attachment:

1. Start a simulation or remote debugging session.
2. In Process Variables view, open a variable that has attachments.
3. Expand Attachments.
4. Right-click on an attachment and select **Remove Attachment.**

To download an attachment:

1. Start a simulation or remote debugging session.
2. In Process Variables view, open a variable that has attachments.
3. Expand Attachments.
4. Right-click on an attachment and select **Download Attachment**.
5. Select a location for the file.

Custom Functions for Manipulating Attachments

In the Expression, Query, and other Builders you can select the following functions to work with attachments. These functions are under the **BPEL > Process Developer > Attachments** category.

Note: Using a SOAP message to send nested attachments is not supported. if the you are sending SOAP messages with multiple attachments that we do not support nested attachments. While testing, you may want to set the SoapUI Disable multiparts property false.

To view usage of the functions, see *Attachment Custom Function Examples*.

abx:base64EncodeAttachment(variableName, attachmentNumber)

Returns the Base64 encoded attachment of the named variable. Base64 refers to a MIME content transfer encoding.

Parameters:

• variableName: Name of the variable containing the attachment

- attachmentNumber: The source variable attachment item number starting at one

abx:getAttachmentCount(variableName)

Returns an integer that is the number of attachment items associated with a named variable.

abx:copyAttachment(fromVariableName, fromItemNumber, toVariableName)

Copies one of the attachments from the source variable to the target variable. The return type is `xs:boolean`. If the copy fails, an attachment fault is thrown.

Parameters

- fromVariableName: Name of the source variable
- fromItemNumber: Source variable attachment item number, starting at one
- toVariableName: Name of the target variable

abx:replaceAttachment(fromVariableName, fromItemNumber, toVariableName, toItemNumber)

Replaces one of the attachments of the target variable with an attachment from the source variable. The return type is `xs:boolean`. If the replacement fails, or an item number is out of range, an attachment fault is thrown.

Parameters:

- fromVariableName: Name of the source variable
- fromItemNumber: Source variable attachment item number, starting at one
- toVariableName: Name of the target variable
- toItemNumber: Target variable attachment item, starting at one

abx:removeAttachment(variableName, itemNumber)

Deletes the attachment of the named variable identified by the toItemNumber. The return type is `xs:boolean`. If the removal fails, or an item number is out of range, an attachment fault is thrown.

Parameters:

- variableName: Name of the variable containing the attachment
- itemNumber: Item number of attachment starting at one

abx:createAttachment(variableName, contentType, encodedContent [, contentId])

Adds the attachment to the named variable specifying the content type and content. The return type is `xs:boolean`. If the creation fails, an attachment fault is thrown.

Parameters:

- variableName: Name of the variable to add the attachment
- contentType:MIME type of the attachment, such as *text/xml* or *image/jpg*
- encodedContent: String that contains Base64 encoded attachment data content
- contentId: (Optional). A sting that refers to the content of the attachment. This parameter can be helpful in accessing a particular attachment when a message has multiple attachments.

abx:copyAllAttachments(fromVariableNames, toVariableName)

Copies all attachments from the list of variables identified by the fromVariableNames (`xsd:string` list) to the variable identified by the toVariableName. The return type is xs:integer. If the copy fails, an attachment fault is thrown.

Parameters:

- fromVariableNames: Names of the variables to copy from. You can enter a single name or multiple name separated with a space. Use * to indicate all in-scope variables.
- toVariableName: Name of the variable to copy to

abx:removeAllAttachments(variableNames)

Deletes all attachments from the list of variables. The return type is `xs:integer`.

Parameter:

- fromVariableNames: Names of the variables to copy from. You can enter a single name or multiple name separated with a space. Use * to indicate all in-scope variables.

abx:getAttachmentType(variableName, itemNumber)

Returns the MIME type of the attachment associated with the named variable.

Parameters:

- variableName: Name of the variable containing the attachment
- itemNumber: Item number of attachment starting at one

abx:getAttachmentProperty(variableName, itemNumber, propertyName)

Returns the value associated with the attachment property.

Parameters:

- variableName: Name of the variable containing the attachment
- itemNumber: Item number of attachment starting at one
- propertyName: Name of the attachment's property

Attachment properties include those specified by the *W3C SOAP Messages with Attachments Specification* (http://www.w3.org/TR/SOAP-attachments), Multipart/Related MIME media type (RFC 2387), and Informatica.

SOAP with Attachments properties include *Content-Type*, *Content-Transfer-Encoding*, *Content-ID*, and *Content-Location*.

Informatica-specified attachment properties include *Attached-By* (the authenticated user who sent the attachment), *X-Size* (size of attachment in bytes), and *Attachment-Created-At* (a server-generated time-value (in milliseconds) of when the attachment was received by the Process Server).

abx:getAttachmentSize(variableName, itemNumber)

Returns the content size for the variable attachment item of the named variable.

Parameters:

- variableName: Name of the variable containing the attachment
- itemNumber: Item number of attachment starting at one

abx:setAttachmentProperty(variableName, itemNumber, propertyName, propertyValue)

Sets and returns the named property of the attachment associated with the named variable.

Parameters:

- variableName: Name of the variable containing the attachment
- itemNumber: Item number of attachment starting at one
- propertyName: Name of the attachment's property
- propertyValue: Value of the attachment's property

This function is particularly useful for giving a BIRT report a meaningful name when the report is a PDF attached to an email.

abx:xmlAttachmentToElement(variableName, attachmentNumber)

Returns an attachment and converts the attachment to an element style variable. The attachment must be an XML document.

Parameter:

- variableName: Name of the variable containing the attachment
- attachmentNumber: The source variable attachment item number starting at one

Attachment Custom Function Examples

The following examples of using custom functions show the manipulation of attachments in copy operations of assign activities. For a detailed description of each function, see *Custom Functions for Manipulating Attachments*.

Example 1:

Copies the first attachment from the `incomingMessage` variable to the `resultMessage` variable. The return value of `copyAttachment` is copied to the unused `tempVar`.

```
<copy>
    <from>abx:copyAttachment
        ("incomingMessage",1,"resultMessage")
    </from>
    <to variable="tempVar"/>
</copy>
```

Example 2:

Removes the (N - 1) attachment; that is, removes the second-to-last attachment from the `resultMessage` variable, where N is the count of attachments on `resultMessage`.The return value of `removeAttachment` is copied to the unused `tempVar`.

```
<copy>
    <from>abx:removeAttachment("resultMessage",
      abx:getAttachmentCount("resultMessage") - 1 )
    </from>
    <to variable="tempVar"/>
</copy>
```

E*xample 3:*

Copies the value of the *Content-Transfer-Encoding* property of the first attachment to the `resultMessage` variable.

```
<copy>
    <from>abx:getAttachmentProperty("resultMessage" , 1 ,
        "Content-Transfer-Encoding")
    </from>
    <to part="propResult" variable="resultMessage"/>
</copy>
```

CHAPTER 11

Using Links

This chapter includes the following topics:

What is a Link

A link is a structure that connects two activities and controls the order of execution of the two activities. In the BPMN-Centric palette, a link is called *edge*.

A link can include a condition, further controlling when, or if, an activity executes. The condition is an XPath (or other language) expression.

You can add multiple links between basic activities and between structured activities. You can also add links to a basic activity contained by a structured activity.

A link works inside of a flow. When you add a link between two activities, Process Developer creates a flow activity including a link definition, as shown.

Link XML Syntax

```
<flow>
 <link name="L1" />
 activity1
 <source linkName="L1" />
 ...
 activity2
 <target linkName="L1" />
 ...
</flow>
```

For more information about when to use links vs. using sequence, scope and other structured activities, see *Designing With Links vs. Structured Activities*.

Process Developer Extension for Links

According to the WS-BPEL 2.0 execution rules for links, a link cannot create a control cycle by linking to a completed activity. In addition, a link cannot cross the boundary of a structured activity, such as a while activity. However, Process Developer includes an extension that allows these behaviors under certain conditions.

The following illustration shows two cases of using loop-back links to a completed activity. On the left is an example of a valid use of a loop-back link. On the right is a case that will cause an error in static analysis because it violates the fork join restriction described below.
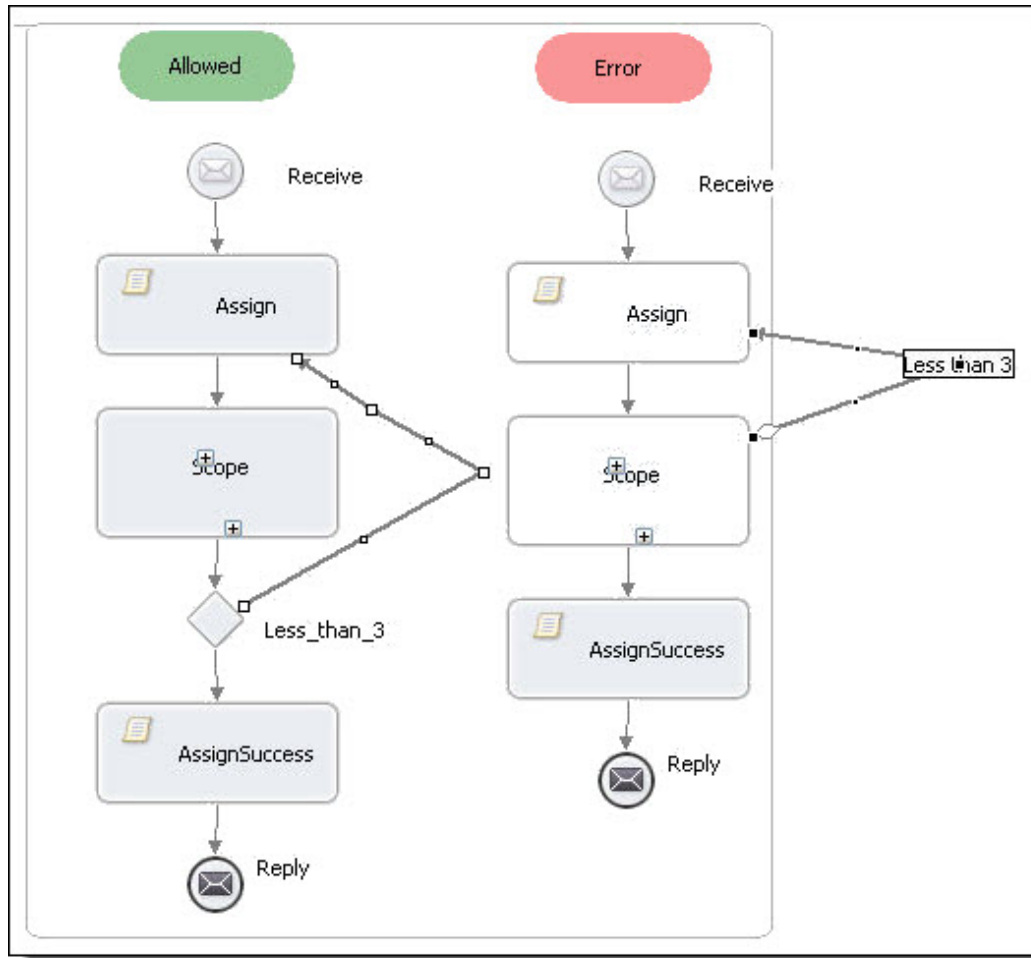
By default, the link extension is turned on, allowing you to use loop-back links. You can turn off the extension on a process-by-process basis by setting the *Links are Transitions* property in the All tab of a process.

Execution Rules for Extension to Links

The basic execution rules for Process Developer extension to links are the following.

- When the source of a link completes, then if the transition condition evaluates to true or there is no condition, the transition fires; if the target of the link is an activity with no other incoming transitions, the activity executes.

- BPEL's constraints regarding how links can cross the boundaries of containers continue to hold except that a Process Developer extension link can cross out of a Repeat Until activity if the source of the link

has the *Mutually Exclusive Transitions* property is set to *Yes* in an activity's Properties view. By default, this property is set to *Yes* for an exclusive gateway, and *No* for all else.

- When an activity has *Mutually Exclusive Transitions* set to *Yes*, the normal control flow of any container that contains the activity is continued only if none of the outbound links from the activity can execute. The behavior should be obvious based on the process diagram since it follows standard BPMN semantics.

BPMN disallows links from crossing the border of any bordered activity. However, BPEL allows links to cross scope boundaries, so Process Developer also allows this behavior, even though it is not valid BPMN.

Usage Restriction for Activities in Parallel Execution

If two or more activities execute in parallel, and you want to create a link that loops back to before the parallelism, the parallelism must be encapsulated within a Fork Join container. This restriction ensures that there will not be concurrent execution of looping activities since the fork join ensures that all activities on parallel paths complete and join together before the next execution of the loop.

We recommend that you use mutually exclusive links as the source links of a gateway activity that is an exclusive type. Drawing links out of an exclusive gateway clarifies the intended behavior of outbound links. Note that using Process Developer extension links as loop-back links is similar to using a While or Repeat Until activity that is already structured for you.

Process Developer Extension Namespace for Links

You may notice that your processes include the Process Developer extension for links in the Extensions node of the Outline.

Adding a Link Between Activities

Add a link between two activities to control the order of execution of the activities. You can add a link with a transition condition or without one.

Adding a Link with no Transition

Add a simple link from a source activity to a target activity to require the source to finish successfully before the target executes.

1. Ensure that two activities are on the Process Editor canvas that you want to link.
2. Select the source activity and then use **Shift + select**to select the target activity.
3. Select the **Link** toolbar button or select **Link Activities** from the right mouse menu of one of the activities.
4. Process Developer names the link `L1`. You can double-click the link to open the **Transition Builder** dialog.
5. In the Properties view, you can rename the link, if desired.

Tip: You can use the BPMN Edge tool in the Palette, if desired. To do so, select the tool and then select the source activity. Connect the link to the target, as shown on the left in the illustration. Select the **Select** tool to turn off the Edge tool.
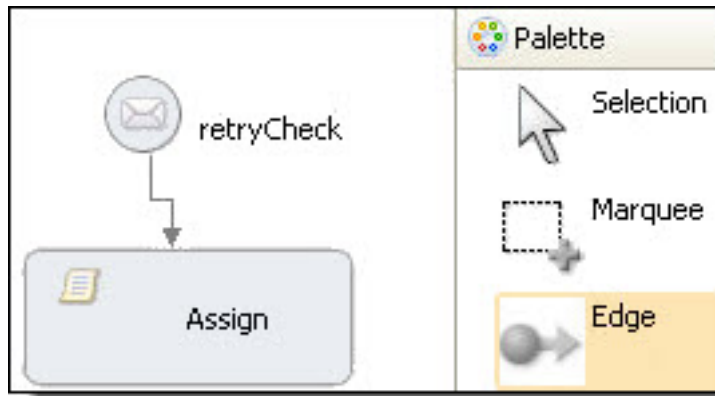
Adding a Link with a Transition Condition

Add a link with a transition condition based on the completion of the source activity. The condition must evaluate to a Boolean true before the target activity can begin.

1. Complete the steps for drawing a link, described in *Adding a Link with no Transition*.

2. Select a link and double-click it. Alternately, in the Properties view, click the `Dialog (...) Button` next to *Transition Condition*.

3. Build an XPath (or other language) expression, as described in *Using the Expression Builder*.
    The following illustration shows an example of condition that prevents an activity from executing if an amount is too large.
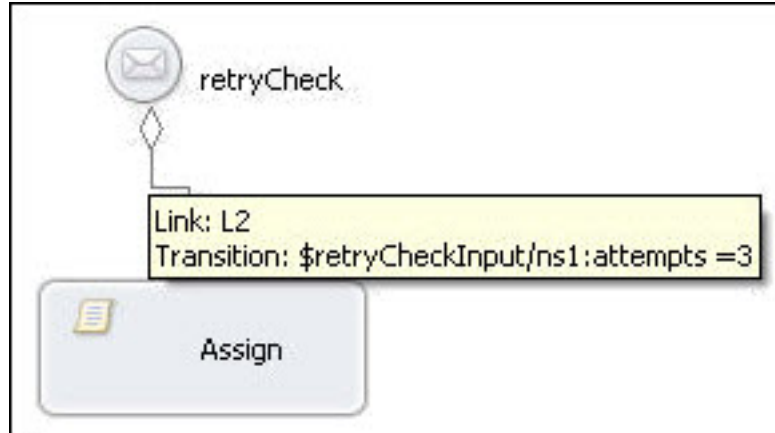
In this example, the parameters refer to a BPEL process variable holding data from a WSDL message. For more information, see *Using Variables*.

Linking Activities

You can link any two activities; however, you should use a gateway to clarify the intended purpose of linking. In the following illustration, the construct on the left shows using a gateway and adding a label to describe the transition condition on the link.

On the right side, is a receive linked to two activities. Note that a transition condition is indicated by a mini-gateway diamond; however, you can add a label here as well.
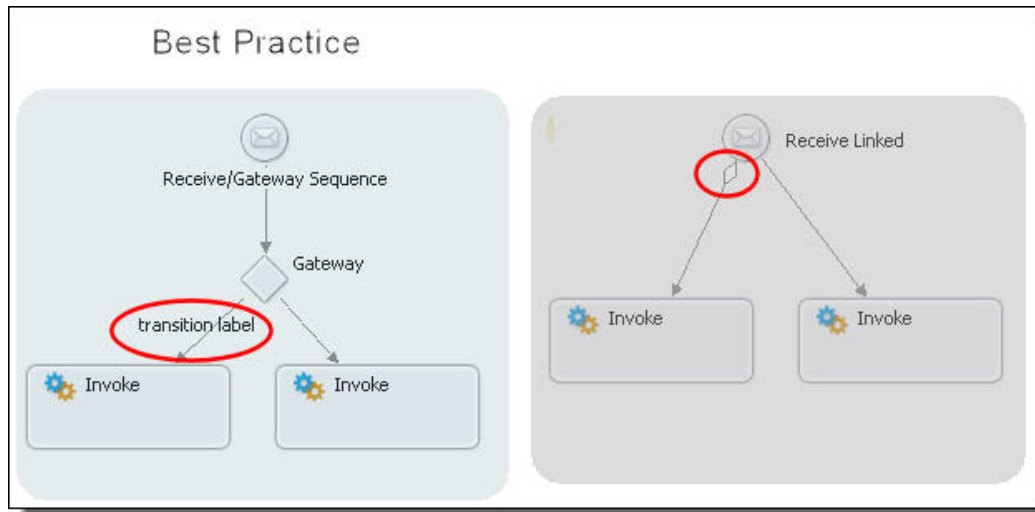
Link Examples

The following illustrations show some ways you can use links.

You can also consider *Designing With Links vs. Structured Activities*.

Example 1

In the first example, the second sequence executes if the link evaluates to true; otherwise, the reply executes.
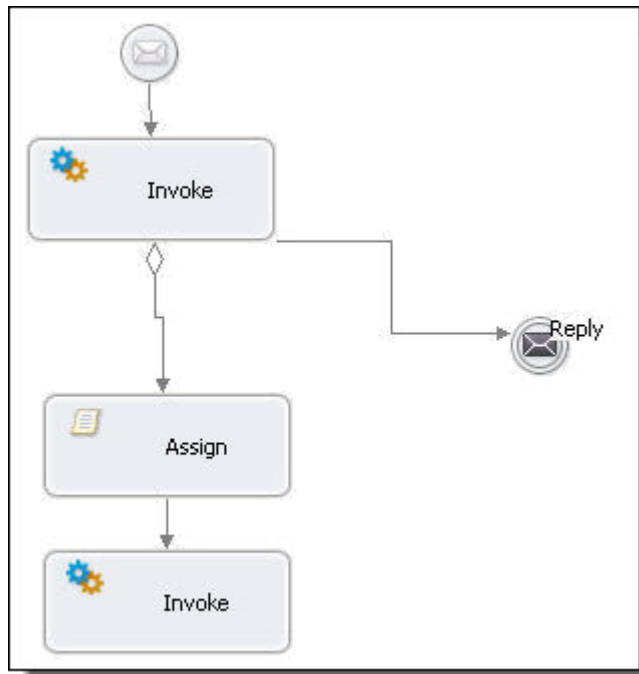
Example 2

In the following example, a scope is running in parallel with a sequence. An activity in one sequence has a link to an activity in the other sequence.
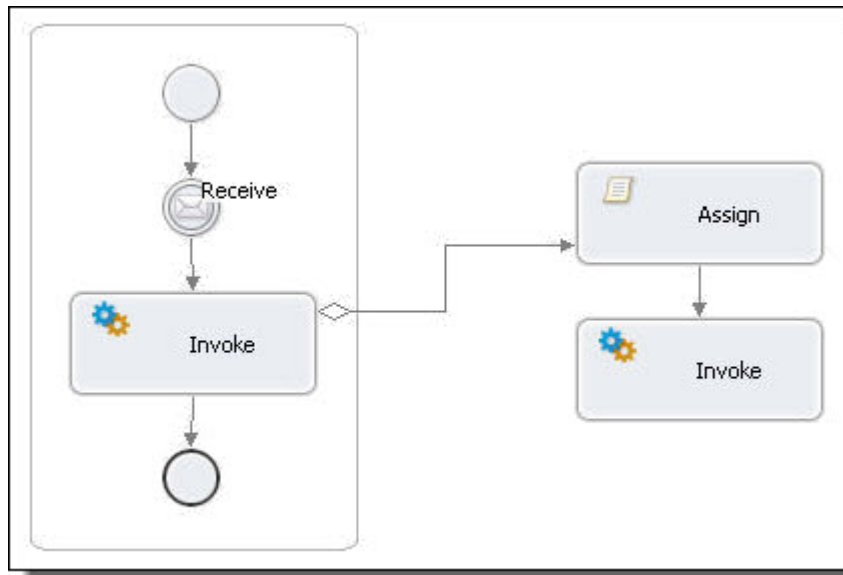
Execution Rules for Links

Once an activity completes, it evaluates any condition on the outgoing links. If no condition is defined and the activity has completed normally, all conditions evaluate to true so that the next activity can start. However, an activity can have a join condition set that further constrains execution. For more information, see *Links and the Join Condition*.

If an activity has faulted or could not be run, all conditions evaluate to false.

An activity can have multiple incoming links. The activity must wait to run until at least one of its incoming links evaluates to true.

If all incoming links are false, the activity cannot run unless a *supressJoinFailure* property is true.

A link can cross the boundary of a structured activity, except for the following:

- While activity
- Isolated scope
- Event handler
- Compensation handler

A link that crosses the boundary of a fault handler must have its source within the fault handler and its target within a scope that encloses the fault handler scope.

In WS-BPEL, a link cannot create a control cycle by linking to a completed activity. However, in Process Developer, this behavior is allowed, as described in *Process Developer Extension for Links*.

Designing With Links vs. Structured Activities

You can often set up the same execution order for activities using either links or a sequence. The following example shows two execution designs that are identical.
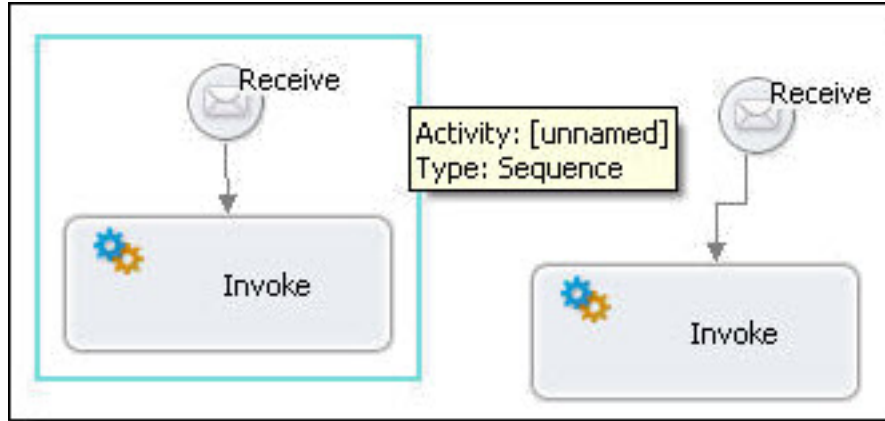
There are many cases when you can design an execution path only with links. For example, you can link an activity from within a container to another activity in a different container.

Links and the Join Condition

You can control the start of an link-targeted activity by writing an expression for a join condition property. For more information, see *Creating a Join Condition for an Incoming Link*.

Link Properties

Select a link on the Process Editor canvas or Outline view to see the Properties view.

The following table shows the required and optional properties for a link.

Required Properties	Optional Properties
Link Name	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images.
	Execution State.
	Extension Attributes and Extension Elements..

CHAPTER 12

Data Manipulation

The following topics tell you how to manipulate XML data in BPEL using literal expressions, XPath, and XQuery.

Overview of Data Manipulation in BPEL

There are many ways to manipulate data in a BPEL process, but it is not always obvious how to perform the data copying tasks easily or efficiently.

Also, you can select an expression language to use on a expression-by-expression basis, depending on your data structure and the selections from it you want to make. For details, see *Selecting XPath or XQuery for Expression Building*.

Selecting XPath or XQuery for Expression Building

The default language is XQuery, which is a superset of XPath. Process Developer projects contain an XQuery nature, including an XQuery editor for writing custom function XQuery modules.

The following table provides a brief comparison between XPath and XQuery as used in BPEL data mapping:

XPath	XQuery
Recommended if portability is an issue. Language is supported by WS-BPEL 2.0	Process Developer extension for BPEL
May be more suitable for multiple-part message variables	Builds large documents easily from a single-part variable
Limited to expression evaluation of one document node at a time	Allows manipulation of data from an entire XML document
	Better at date handling, with many date functions

XPath	XQuery
Location path expressions resolve to a string	Queries generate an XML result
	Similar to a SQL-like query language. Easier to iterate over repeating elements with "FLWOR expressions" (FOR, LET, WHERE, ORDER BY, and RETURN) for performing joins

See also:

-
-

Example XQuery Expressions

If you select the XQuery expression language, you can use the many functions that are supported by XQuery in addition to all XPath expression building. The additional XQuery function categories include Date, QName, Misc, and Constructor.

You can then create expressions for the document, as the following example shows:

Note the syntax used to generate the contents of an element, such as the following from the example above:

```
<aem:replyTo>{$runtimeParameters/param:emailAddress/text()}</aem:replyTo>
```

You must add the `text()` function to the end of the path. Any time an XQuery expression results in an element, you must add `text()` to get the contents of the element.

The path without `/text()` would result in the following:

```
<aem:replyTo>
    <emailAddress>reply@example.org</emailAddress>
</aem:replyTo>
```

instead of the desired result:

```
<aem:replyTo>reply@example.org</aem:replyTo>
```

If the expression contains just a simple value and not an XML element, the expression results to the contents. For example, the following expression results in a string:

```
Please use the Claim #: {$refNumber}
```

Example XPath Expressions

The following are example expressions for message, element and complex schema type variables.

- **Element type variable:**
  Example: `$AStatus/e:statusDescription`

  Selects the `statusDescription` child node of `AStatus` element

- **Message type variable:**
  Example: `$StatusVariable.StatusPart2/e:statusDescription`

  Selects the `statusDescription` child node of the `StatusPart2` part of `StatusVariable`

- **Schema complex type variable:**
  Example: `$Results/e:AuctionResult[2]/@AuctionID`

  Selects the `AuctionID` attribute of the second `AuctionResult` child node

- **Expression statement** `itemsShipped = itemsShipped + itemsCount`:
  `$itemsShipped + $shipnotice.props/itemsCount`

- **Convert** a variable and part to a string:
  ```
  string($input.parameters)
  ```

- **Referencing a child node by localname only**
  This can be useful when a schema has an `xsd:any` element in it, such as:

  ```
  <xs:any minOccurs="0" maxOccurs="unbounded"        namespace="##other"
  processContents="lax"/>
  ```

The Expression Builder doesn't show the structure in the variable tree view. To use the element, the namespace prefix can be ignored as in this example:

```
$creditInformation/*[local-name()=
   "previousLoanApplicationAttempts"][1]
   /*[local-name()="applicationDate"]
```

Selects the date of the first previous loan application, if information about previous loan applications is supplied as additional content for the creditInformation message.

Using the Expression Builder

Select from the Variables, Functions, and Operators pick lists to create an expression in the Expression/ Condition text box.

You can select among the many function types to create an expression for Copy Operations, link transitions, and other BPEL constructs. See the procedure below for details on building an expression.

See also *Selecting XPath or XQuery for Expression Buildin*g.

Examples

- Expected Expressions for Conditions, Counters, and Other Values

- Example XPath Expressions

- Example XQuery Expressions

Functions

- Boolean Functions

- BPEL Functions

- Node Set Functions

- Number Functions

- Process Developer Custom Functions: General

- String Functions

See also the BPEL for People Functions in *Process Developer Human Tasks Help*

Customization

- Custom Functions Overview

- Writing XQuery Functions

- Select Expression Language

For Copy Operations in the Assign activity as well as for links and other constructs, you can create XPath (or another selected language) expressions that result in a string, number, or Boolean.

For details, see the XPath 1.0 specification at `http://www.w3.org/TR/xpath`.

To build an expression:

1. Select an activity that uses expressions, such as the assign, wait, or on alarm branch of a pick activity.

2. In the Properties view for the activity, select the appropriate property. For example, in an assign activity, select Copy Operations, and select Expression as a Copy/From type to display the following dialog:
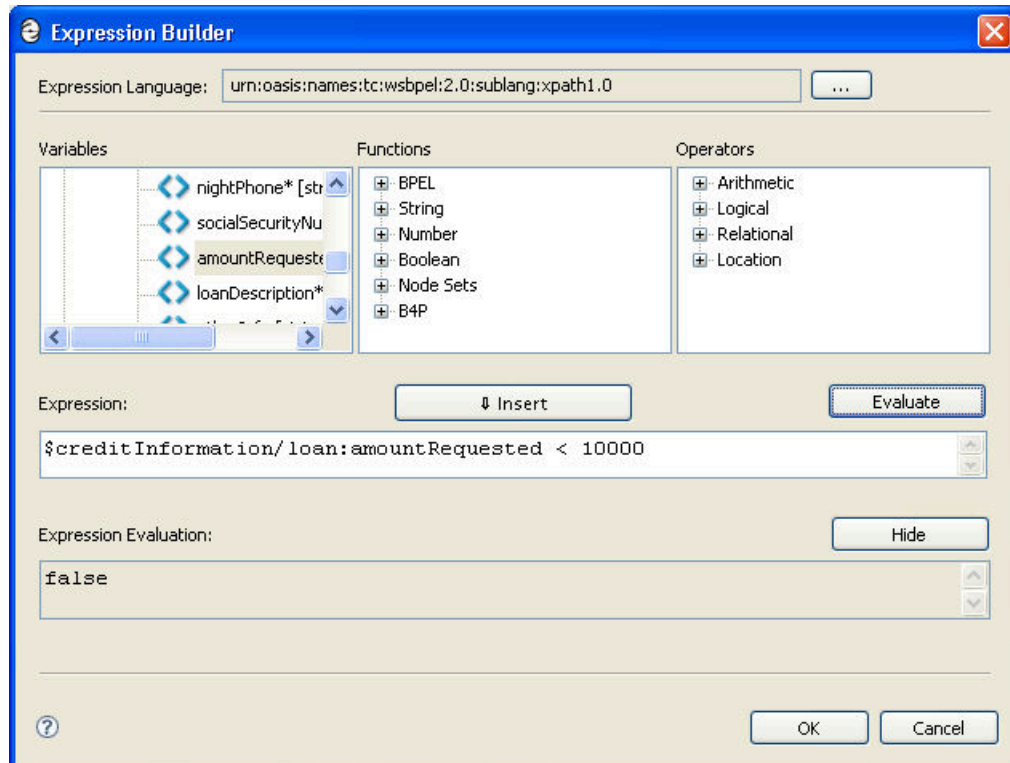
3. Select the variables, functions and operators from the three boxes to build an expression, following these tips:

   - Build expressions faster with shortcuts. See *Using Content Assist*.

   - Expand a variable to see and use the parts and properties. For detailed descriptions of variables, see *Using Variables*.

   - Select a variable, part, function, or operator and select **Insert**

   - For working with variable attachments, see *Custom Functions for Manipulating Attachments*.

   - Add your own custom functions, as described in *Creating POJO and XQuery Custom Functions*.

4. Select **Evaluate** to substitute a sample value for a variable part and simplify the expression to view its value.

Using Content Assist

Select from the Variables, Functions, and Operators pick lists to create an expression in the Expression/ Condition text box.

Content assist is a shortcut for making selections from the variables and functions available for building an expression. Instead of expanding a variable to drill-down to a part to insert into the expression text box, you can select the part within the expression text box.

To use content assist:

1. In the builder (expression, query, or other), place your cursor in the expression text box, located below the Insert button.

2. To select a function, select Ctrl + spacebar. A list of functions appears that you can select from.

3. To select a variable, part or element, select Ctrl + spacebar as follows:

    a. If the cursor is after a $, a list of available variables appears, if applicable.

    b. If the cursor is after a period (.), and if the context is in an XPath expression, a list of valid part names appears, if applicable.

    c. If the cursor is after a slash (/), and if the context is in an XPath expression, a list of valid element names appears, if applicable.

BPEL Functions

This section describes the following functions:

-
-

bpel doXslTransform(style-sheet-uri node-set)

Returns the result of an XSL transformation of a single element node set using the specified style sheet.

Parameters:

- `style-sheet-uri`: URI for the XSL file.

- `node-set`: XPath node set providing the source document for the transformation.

Example syntax:

```
bpel:doXslTransform("project:/myStylesheets/A2B.xsl", $A)
```

Optional parameters (must appear in pairs if specified)

- `string`: XPath string parameter providing the qualified name of an XSLT parameter.

- `object`: XPath object parameter providing the value for the named XSLT parameter.

Examples

The following examples show complex document transformation and iterative document construction.

Example 1. Complex Document Transformation.

A common pattern in a WS-BPEL process involves receiving an XML document from one service, converting it to a different schema to form a new request message, and sending the new request to another service. Such document conversion can be accomplished using XSLT via the `bpel:doXslTransform` function, as shown in the following example.

```
<variables>
    <variable name="A" element="foo:AElement" />
    <variable name="B" element="bar:BElement" />
</variables>
...
<sequence>
    <invoke ... inputVariable="..." outputVariable="A" />
    <assign>
      <from>
          bpel:doXslTransform("urn:stylesheets:A2B.xsl", $A)
      </from>
      <to variable="B" />
    </assign>
```

```
      <invoke ... inputVariable="B" ... />
   </sequence>
```

In the sequence, a service is invoked, and the result (`foo:AElement`) copied to variable A. The assign activity transforms the contents of variable A to `bar:BElement`, and copies the result to variable B. Variable B is used to invoke another service. The style sheet, `A2B.xsl`, contains the XSL rules for converting documents of schema `foo:AElement` to schema *bar:BElement*.

Example 2. Iterative Document Construction.

Suppose that a document is constructed by repeatedly calling a service and accumulating the result in a variable, as shown in the following example:

```
<variables>
    <variable name="PO" element="foo:POElement" />
    <variable name="OutVar" element="foo:ItemElement" />
</variables>
<!-- ... PO is initialized ... -->
<!-- Iteratively add more items to PO until complete -->
<while>
    <condition>...</condition>
     <sequence>
        <!-- Fetch next chunk into OutVar -->
      <invoke ... inputVariable="..." outputVariable="OutVar"/>
       <assign>
          <copy>
           <from>
              <expression>
                 bpel:doXslTransform
                   ( "urn:stylesheets:AddToPO.xsl",
                      $PO, "NewItem", $OutVar)
              </expression>
           </from>
           <to variable="PO" />
          </copy>
       </assign>
     </sequence>
</while>
```

The optional parameters given in the `doXslTransform` call specify that the XSLT parameter named `NewItem` is set with the value of the process variable `OutVar`. To allow the XSLT style sheet access to this value, it contains a global (top-level) parameter with a name matching that given in the third parameter of the function call shown above.

```
<xsl:transform version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ...>
     <!-- NewItem variable set by WS-BPEL process;
        defaults to empty item -->
    <xsl:param name="NewItem">
        <foo:itemElement />
    </xsl:param>
        ...
</xsl:transform>
```

The style sheet contains a template that appends the value of global parameter NewItem (the value of `OutVar` from the process instance) to the existing list of items in the PO variable.

```
<!-- line 1 -->  <xsl:template match="foo:itemElement">
<!-- line 2 -->  <xsl:copy-of select="." />
<!-- line 3 -->    <xsl:if test="position()=last()">
<!-- line 4 -->      <xsl:copy-of select="$NewItem" />
<!-- line 5 -->    </xsl:if>
<!-- line 6 -->  </xsl:template>
```

This template copies all existing items in the source document (lines 1 & 2) and appends the contents of the XSLT parameter `NewItem` to the list of items (lines 3 & 4). It tests to see if the current node is at the end of the item list (line 3) and copies the `result-tree` fragment from the XSLT parameter `NewItem` to follow the last item (line 4).

If PO has a value of:

```
<foo:poElement>
      <foo:itemElement>item 1</foo:itemElement>
</foo:poElement>
```

at the beginning of an iteration of the while loop and the invoke activity returns a value of `<foo:itemElement>item 2</foo:itemElement>`, evaluation of the from expression will result in a value of:

```
<foo:poElement>
      <foo:itemElement>item 1</foo:itemElement>
      <foo:itemElement>item 2</foo:itemElement>
</foo:poElement>
```

When the copy operation completes, it becomes the new value of the PO variable.

Adding an XSL Style Sheet to Process Server

A style sheet that you reference in a `doXslTransform` function is automatically recognized by Process Developer and is added to a BPR archive for deployment.

bpel getVariableProperty(VariableName propertyName)

Returns property data from a selected variable in a BPEL process.

Parameters:

- `VariableName`: The name of the variable to retrieve data from
- `propertyName`: the name of the property in the variable that contains the data

Process Developer Custom Functions General

The following topics describe Process Developer custom functions:

abx base64Encode(stringValue charSet)

Returns the Base64 encoded string.

Parameters:

- `stringValue`: The string to be encoded
- `charSet`: The optional named character set to use for encoding. The default is UTF-8.

abx elementToXMLString(element)

Returns an XML string given an `element`.

abx getAllHTTPHeaders(partnerLinkName)

Returns a map of HTTP header values from the last message received on the PartnerLink in the business process. Applies to any type of inbound HTTP request without regard to whether the payload is SOAP, XML, or JSON.

Parameter:

- `partnerLinkName`: The name of the PartnerLink to retrieve data from

abx getHTTPHeader(partnerLinkName headerName)

Returns the value of any HTTP header from the last message received on the PartnerLink in the business process. Applies to any type of inbound HTTP request without regard to whether the payload is SOAP, XML, or JSON.

Parameters:

- `partnerLinkName`: The name of the PartnerLink to retrieve data from.
- `headerName`: The name of the header

abx getInboundSOAPHeader(partnerLinkName)

Returns SOAP message header block from the last message received on the PartnerLink in the business process.

Parameter:

- `partnerLinkName`: The name of the partner link to retrieve data from

abx getProcessId()

Returns the process Id of the currently executing or completed process on the Process Server.

abx getProcessInitiator() (1)

Returns the principal attached to the createInstance message or anonymous if there are no credentials or principal associated with the message.

abx getProcessName()

Returns the process name (local part only) of the currently executing process.

abx isVariableInitialized(variable)

Returns true if `variable` is initialized.

abx resolveURN(string)

Returns the resolved location of the URN using the URN/URL mapping facility of the server. Takes a single string parameter as input. For example, `abx:resolveURN('urn:localhost:AssessRisk')`

abx setProcessTitle(titleString)

Set the display title of an active process on the Active Processes page of the Process Console. Does not replace the process name.

abx xmlStringToElement(string)

Returns the Element given an XML `string`.

Attachment Functions

These functions are described in <u>"Custom Functions for Manipulating Attachments" on page 226</u>.

Boolean Functions

You can use the following Boolean functions:

- <u>"boolean(object) " on page 246</u>
- <u>"not(boolean) " on page 246</u>
- <u>"true() " on page 246</u>
- <u>"false() " on page 246</u>
- <u>"lang(string) " on page 246</u>

boolean(object)

The Boolean function converts its argument to a Boolean as follows:

- A number is true if and only if it is neither positive or negative zero nor NaN
- A node-set is true if and only if it is non-empty
- A string is true if and only if its length is non-zero

Parameter:

- `object`: An object of a type other than the four basic types is converted to a Boolean in a way that is dependent on that type.

not(boolean)

Returns true if its argument is false, and false otherwise.

true()

Returns true.

false()

Returns false.

lang(string)

The `lang` function returns true or false depending on whether the language of the context node as specified by `xml:lang` attributes is the same as or is a sub-language of the language specified by the argument string. The language of the context node is determined by the value of the `xml:lang` attribute on the context node,

or, if the context node has no `xml:lang` attribute, by the value of the `xml:lang` attribute on the nearest ancestor of the context node that has an `xml:lang` attribute. If there is no such attribute, then `lang` returns false. If there is such an attribute, then `lang` returns true if the attribute value is equal to the argument ignoring case, or if there is some suffix starting with - such that the attribute value is equal to the argument ignoring that suffix of the attribute value and ignoring case.

Catalog Functions

These custom functions are capable of loading any resource out of the Process Server catalog and updating a resource in the catalog. The loaded resource is returned to the expression language as an XML document. For example, you can extract configuration information from process definition, such as alarms or endpoint references, and keep it in the catalog where it can be changed and shared across process definitions. The use of the catalog also enables a process to have different configuration information based on its deployment environment. For example, a development environment can have one type of configuration information while a production environment can have another. The configuration information is in XML format. Be sure to import the resource XML files into the project to make them available for simulation and for automatic export to a BPR archive. For details, see "Importing WSDL Schema and Other Resources" on page 100 and "Creating and Deploying a Business Process Archive Contribution " on page 372.

The catalog functions you can use are:

- "abx getCatalogResource(location) " on page 247
- "abx putCatalogResource(location typeURI document) " on page 248

abx getCatalogResource(location)

Returns the document containing the resource content. The location is the string containing the catalog location URL. The example below returns the element or document that was stored using `putCatalogResource()`, by the location. You can use this function in a copy operation to assign the element back to a variable, for example.

If the request header uses `application/json` or the resources ends with `.tojson` (for example, `/avccatalog/project:/path/mydoc.xml.tojson`), you do not have to write a process to proxy the request if the endpoint is `/avccatalog`. Also, if the request header has `application/json`, you do not have to write a proxy if the endpoint is `/catalog`.

Example:

```
abx:getCatalogResource("project:/myProject/resources/resource.xml")
```

In a copy operation, an example of copying From this function (to a variable) is as follows. Note that the expression language is XQuery.

```
<ns1:catalogEntry xmlns:ns1='http://activevos/samples/catalog/functions'>
  <ns1:catalogLocation>
    <ns1:location>{data(
     $requestOperation.catalogOp/
     ns1:catalogLocation/ns1:location )
    }</ns1:location>
    <ns1:typeURI>{data( $requestOperation.catalogOp/
        ns1:catalogLocation/ns1:typeURI )}</ns1:typeURI>
  </ns1:catalogLocation>
  <ns1:content>{abx:getCatalogResource( data(
      $requestOperation.catalogOp/ns1:
      catalogLocation/ns1:location))}</ns1:content>
</ns1:catalogEntry>
```

abx putCatalogResource(location typeURI document)

Updates or adds the catalog resource at the passed location. This function returns a Boolean so you can use it in the From side of a copy operation and copy it to a variable of type Boolean. The example below puts the document or element in variable `$V1` in the catalog with the identifier `project:/myProject/resources/resource.xml` and defined as type `http://www.example.org/2009/03/`. If the type is a WSDL, you can use the WSDL namespace. If you do not specify the type, then the namespace of the element for `$V1` is automatically used.

Example:

```
abx:putCatalogResource(    "project:/myProject/resources/resource.xml" ,    "http://
www.example.org/2009/03/", $V1)
```

In a copy operation, an example of copying from this function (to a Boolean variable) is:

From:

```
abx:putCatalogResource( string($requestOperation.catalogOp/ns1:catalogLocation/
ns1:location/text()), $requestOperation.catalogOp/ns1:content/*)
```

To:

```
funcResponse
```

Parameters:

- `location`: The string containing the catalog location url. The location is a hint
- `typeURI`: Optional string containing the resource namespace type (element namespace used if not passed).
- `document`: The xml content (document or element) to be put in the catalog resource.

The function returns a Boolean true or false value.

Fault Functions

The Fault custom functions are useful in cases where partner services have not properly implemented their interfaces by declaring fault messages. Some WSDL operations do not declare a fault message, and when the service throws a fault, it cannot be caught in a normal way, by a catch or catchAll handler. Use these functions to catch undeclared faults. Use the functions within a process variable within an activity triggered by a catch or catchAll fault handler. For example: via an XPath expression `$soapFault/faultstring/text()`.

See example of a returned value below the following faults.

The fault functions you can use are:

- "getFaultCode() " on page 248
- "getFaultDetail() " on page 248
- "getFaultString() " on page 249
- "getSOAPFault() " on page 249

getFaultCode()

Returns Fault Code `QName`, if any, from the event that triggered the fault handler

getFaultDetail()

Returns Fault Detail element, if any, from the event that triggered the fault handler

getFaultString()

Returns Fault message string, if any, from the event that triggered the fault handler.

Here is an example of a returned fault message:

```
<SOAP-ENV:Fault xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <faultcode>SOAP-ENV:Server</faultcode>
    <faultstring>the_fault_string</faultstring>
    <detail/>
</SOAP-ENV:Fault>
```

getSOAPFault()

Returns SOAP Fault element, if any, from the event that triggered the fault handler

I18N Functions

getLocalizedMessage(key, locationHint, locale, [arg1, argN]

Returns the value of a key for an externalized string in a specified locale. The key is from a message properties file, created for Process Central process request forms. Use this function in a REST-based process to send localized data back to the output message of a process request form. In a REST process, the browser locale is available, allowing you to localize returned data.

Parameters:

- `key`: required string - the key name from the message properties file created in Process Developer. The value for the key is returned by the function.
- `locationHint`: required string - The catalog location URL for the default properties file. The location is a hint.
- `locale`: required string - the locale for the message, such as `en_US` or `fr_CA` or `fr`. Empty string for the default locale. The locale is based on the browser locale.
- `arg1`: optional string or object - Argument to replace text within the key value string
- `argN`: optional string or object - Same as `arg1`, the number of args will vary depending on how many placeholders are in the key value

Example:

```
abx:getLocalizedMessage('loan.description','project:/myLoanApprovalProject/deploy/
messages.properties', 'fr', $loanAmount)
```

Note that the `arg1`, `argN` object syntax is based on Java `messageformat`.

JSON Functions

Use these JavaScript Object Notation (JSON) functions in conjunction with REST endpoints, where the request and or the response can be JSON. For details see <u>"Using a REST-based Service " on page 408</u>.

The JSON functions you can use are:

- <u>"jsonToXml(jsonString) " on page 249</u>
- <u>"xmlToJson(element) " on page 250</u>

jsonToXml(jsonString)

Convert the JSON string to an XML element. This string can represent for an array.

The following examples show how four different array representations are transformed into XML:

- The root element and the value is a JSON object.
  JSON:

  ```
  {"foo":{"name":"foo","desc":"foo desc"}}
  ```

  XML:

  ```
  <foo desc="foo desc" name="foo"/>
  ```

- The root element and the value is an array. A wrapper object element containing a sequence of elements representing the values of the array is created.
  JSON:

  ```
  {"foo":[{"name":"foo","desc":"foo desc"},{"name":"foo2","desc":"foo2 desc"}]}
  ```

  XML:

  ```
  <json:object xmlns:json="http://schemas.activebpel.org/JSON/2013/10/10/aeJSON.xsd">
     <foo desc="foo desc" name="foo"/>
     <foo desc="foo2 desc" name="foo2"/>
  </json:object>
  ```

- A root object exists but there is no root element. The JSON object is wrapped into an object element.
  JSON:

  ```
  {"name":"foo","desc":"foo desc"}
  ```

  XML:

  ```
  <json:object xmlns:json="http://schemas.activebpel.org/JSON/2013/10/10/aeJSON.xsd"
     desc="foo desc"
     name="foo"/>
  ```

- A root array exists but there is no root element. The array is wrapped in an array element and each object becomes a sequence of object elements.
  JSON:

  ```
  [{"name":"foo","desc":"foo desc"},{"name":"foo2","desc":"foo2 desc"}]
  ```

  XML:

  ```
  <json:array xmlns:json="http://schemas.activebpel.org/JSON/2013/10/10/aeJSON.xsd">
     <json:object desc="foo desc"  name="foo" />
     <json:object desc="foo2 desc" name="foo2" />
  </json:array>
  ```

xmlToJson(element)

Returns the `element` as a JSON string. The element is the XML document or element to convert.

Node Set Functions

You can use the following node set functions:

count(node-set)

Returns the number of nodes in the argument `node-set`.

id(object)

The `id` function selects elements by their unique ID. When the argument to `id` is of type node-set, then the result is the union of the result of applying id to the string-value of each of the nodes in the argument node-set. When the argument to `id` is of any other type, the argument is converted to a string as if by a call to the string function; the string is split into a whitespace-separated list of tokens (whitespace is any sequence of characters matching the production S); the result is a node-set containing the elements in the same document as the context node that have a unique id equal to any of the tokens in the list.

last()

Returns a number equal to the context size from the expression evaluation context.

local-name(node-set)

The local-name function returns the local part of the expanded-name of the node in the argument `node-set` that is first in document order. If the argument `node-set` is empty or the first node has no expanded-name, an empty string is returned. If the argument is omitted, it defaults to a `node-set` with the context node as its only member.

name(node-set)

The name function returns a string containing a QName representing the expanded name of the node in the argument `node-set` that is first in document order. The QName must represent the expanded-name with respect to the namespace declarations in effect on the node whose expanded-name is being represented. Typically, this is the QName that occurred in the XML source. This need not be the case if there are namespace declarations in effect on the node that associate multiple prefixes with the same namespace. However, an implementation can include information about the original prefix in its representation of nodes; in this case, an implementation can ensure that the returned string is always the same as the QName used in the XML source. If the argument `node-set` is empty or the first node has no expanded name, an empty string is returned. If the argument is omitted, it defaults to a `node-set` with the context node as its only member.

namespace-uri(node-set)

The namespace-uri function returns the namespace URI of the expanded-name of the node in the argument `node-set` that is first in document order. If the argument `node-set` is empty, the first node has no expanded-name, or the namespace URI of the expanded name is null, an empty string is returned. If the argument is omitted, it defaults to a `node-set` with the context node as its only member.

position()

Returns a number equal to the context position from the expression evaluation context.

Number Functions

The number functions that you can use are:

-
-

ceiling(number)

Returns the smallest (closest to negative infinity) `number` that is not less than the argument and that is an integer.

floor(number)

Returns the largest (closest to positive infinity) `number` that is not greater than the argument and that is an integer.

number(object)

Converts `object` to a number.

round(number)

Returns the `number` that is closest to the argument and that is an integer.

- If there are two such numbers, then the one that is closest to positive infinity is returned.
- If the argument is NaN, then NaN is returned.
- If the argument is positive infinity, then positive infinity is returned.
- If the argument is positive zero, then positive zero is returned.
- If the argument is negative infinity, then negative infinity is returned.
- If the argument is negative zero, then negative zero is returned.
- If the argument is less than zero, but greater than or equal to -0.5, then negative zero is returned.

sum(node-set)

Returns the sum, for each node in the argument node set, of the result of converting the string-values of the node to a number.

String Functions

The string functions that you can use are:

concat(string1 string2 string3...)

Concatenates the passed strings. Can be passed any number of strings to concatenate.

normalize-space(string)

Returns the passed string with the leading and trailing spaces trimmed off.

contains(string string)

Returns true if the first string contains the second string.

starts-with(mainString lookForString)

Returns true if the first string starts with the second string.

string(object)

Converts the passed `object` to a string.

string-length(string)

Returns the length of the passed string.

substring(string number number)

Returns the part of the string starting at the first number (the first character being 1) and optionally for the length indicated by the second number.

substring-after(string string)

Returns the first part of the string that occurs after the first occurrence of the second string

substring-before(string string)

Returns the part of the first string that is before the first occurrence of the second string.

translate(string string string)

Returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string. For example, `translate("bar","abc","ABC")` returns the string BAr. If there is a character in the second argument string with no character at a corresponding position in the third argument string (because the second argument string is longer than the third argument string), then occurrences of that character in the first argument string are removed. For example, `translate("--aaa--","abc-","ABC")` returns "AAA". If a character occurs more than once in the second argument string, then the first occurrence determines the replacement character. If the third argument string is longer than the second argument string, then excess characters are ignored.

Expected Expressions for Conditions Counters and Other Values

The following table shows expected expressions for conditions, counters, and other values.

Expression Type	Expected Value
Assign Copy From/To Expression	Any valid from-spec expression
Assign Copy From/To Query	Any valid from-spec expression containing a message part and location
For Each Completion Condition Expression	unsigned integer used to define condition of N out of M
For Each Start Counter Value	unsigned integer
For Each Final Counter Value	unsigned integer
If Condition If Expression	Boolean
Else If Condition Else If Expression	Boolean
Link Transition Condition	Link status (Boolean)
Event Handler On Alarm Deadline	For correctly formatted values, see "Deadline and Duration Expressions " on page 257
Event Handler On Alarm Duration	For correctly formatted values, see "Deadline and Duration Expressions " on page 257
Join Condition	Boolean
Repeat Until Condition	Boolean
Wait Deadline	For correctly formatted values, see "Deadline and Duration Expressions " on page 257
Wait Duration	For correctly formatted values, see "Deadline and Duration Expressions " on page 257
While Condition	Boolean
Variable initialization expression or query	Any valid from-spec expression

Using the Query Builder

Select from the Part, Variables, Functions, and Operators pick lists to create a query in the Query text box. Select an expression language to use for the current expression.

A copy operation in an assign activity can contain an XPath (or other selected language) query attribute that results in a string, number, or Boolean. The value of the query attribute is a query string that identifies a single value within a source or target variable part.

The Query Builder dialog is similar to the Expression Builder dialog, with the addition of a Variable Part tree for easy selection of parts.

For details on XPath syntax, see the XPath specification at `http://www.w3.org/TR/xpath`.

To build a query:

1. In the Properties view for an assign activity, select Copy operations.

2. In the Copy Operations dialog, select Variable from the Type field.

3. If desired, for easier query building, select a Part.

4. Click **Query Builder**.
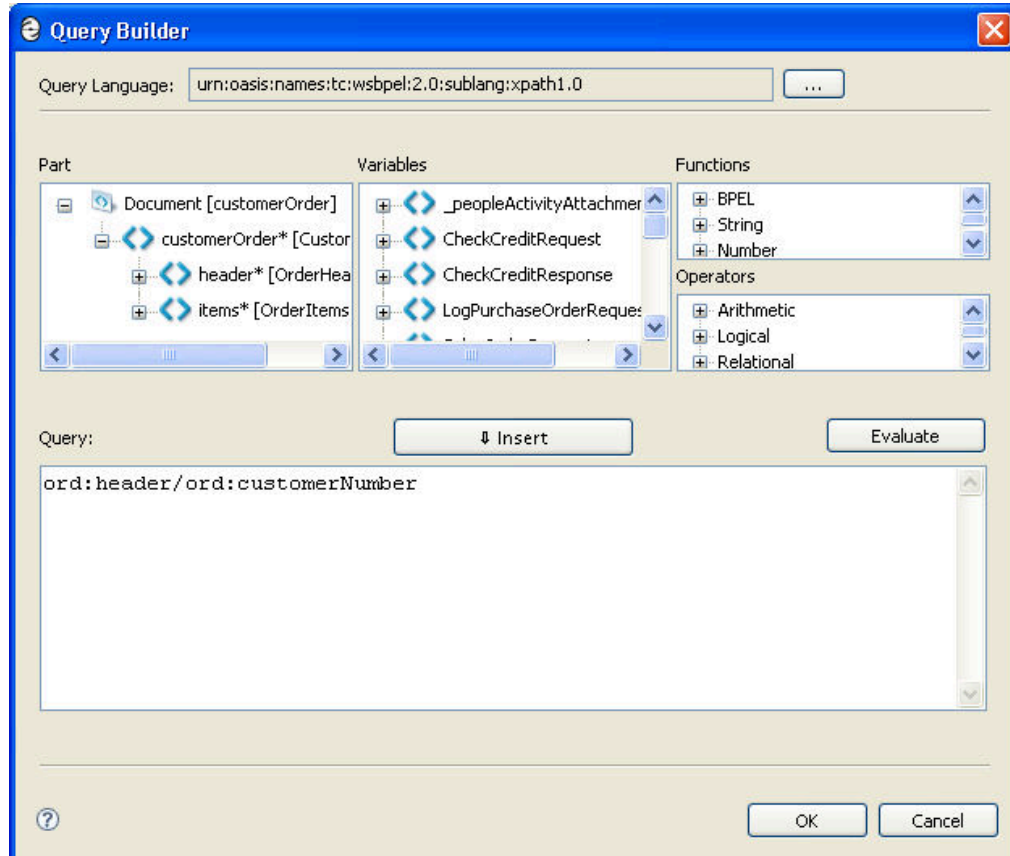
5. If needed, select the `Dialog (...) Button` in the Query Language field to select a different query language URI that non-Process Developer engines support. Do the following:

   • In the Query Language dialog (or in the Properties view of the BPEL process), type in a different URI for a non-Process Developer configured language. The URI is ignored by Process Developer.

   • To restore the default, clear the URI and click **OK**.

6. Select the variable parts, functions and operators from the boxes to build a query, following these tips:

   • For a shortcut, use content assist, as described in "Using Content Assist" on page 241

   • Select a variable, part, function, or operator and select **Insert**

   • Double-click a variable part to generate the location information for the query

   • Double-click a variable to paste the `$variableReference` function into the Query box

   • Double-click on a branch under a part to paste the location into the Query box

- Add custom functions, as described in "Process Developer Custom Functions General " on page 244
- Add your own custom functions, as described in Chapter 28, "Creating POJO and XQuery Custom Functions" on page 400

7. Select **Evaluate** to substitute a sample value for a variable part and simplify the expression to view its value.

Creating a Join Condition for an Incoming Link

Set a join condition to evaluate the success or failure of incoming links. A join condition is a Boolean expression indicating the status of a link targeting an activity.

When one or more links are connected to an activity, you can set a join condition to evaluate the success or failure of the incoming links. A join condition is a property of all activity types and allows you to construct complex execution conditions based on link status. For more information, see *Using Links*.

When the join condition evaluates to true, the process continues normally. When a join condition evaluates to false, the process can throw a `joinFailure` fault.

If you set the *suppressJoinFailure* property to `yes` for an activity that has a join condition, it means that a false join condition will not throw a `joinFailure` fault.

To create a join condition:

1. Display the Properties view of an activity with an incoming link.

2. Select the **Show Advanced Properties** button to display all properties of the activity.

3. Click the `Dialog (...) Button` next to Join Condition to display the following dialog.
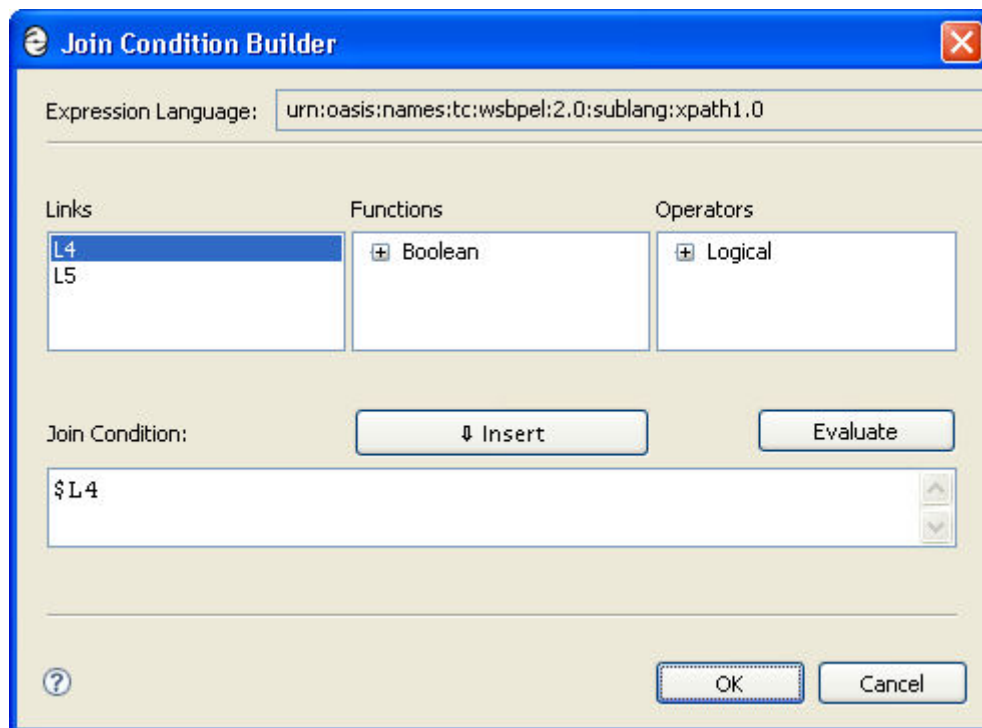
4. Double-click a link to paste the BPEL link status function into the Join Condition box.

This function returns a Boolean indicating the status of the passed link.

5. Click **OK**.

Deadline and Duration Expressions

Several activities contain properties to set a deadline or duration. For deadline and duration expressions, you can see the examples given in the XML schema specification.

Be sure to enclose deadline and duration values in single quotes, as shown in the examples below.

Refer to the following:

- `http://www.w3.org/TR/xmlschema-2/#duration`

- `http://www.w3.org/TR/xmlschema-2/#datetime`

- `http://www.w3.org/TR/xmlschema-2/#date`

Also see the format options described in .

Examples:

- A duration of one second:
  ```
  <wait for="'PT1S'"/>
  ```

- A duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes:
  ```
  <wait for= " 'P1Y2M3DT10H30M' "/>
  ```

- A deadline with a date and time:
  ```
  <wait until=" '2010-12-12T12:00'"/>
  ```

CHAPTER 13

Compensation

This chapter includes the following topics:

Compensation Handlers and Compensate Activities

Compensation Handlers and Compensate Activities

You can add compensation handling as follows:

- Define a compensation handler for a scope. The compensation handler can only have a single child activity, but this activity can be a structured activity like a sequence that has child activities. The contents of the compensation handler depend on your application logic.

- Use a compensate activity within a fault handler, compensation handler, or termination handler to specify compensation on all inner scopes that have already completed successfully, in default order.

- Use a compensate scope activity within a fault handler, compensation handler, or termination handler to specify compensation for a named, inner scope. This method provides control over the order and selection of compensation activities in a scope.

XML Syntax

```
<compensationHandler>
    activity
</compensationHandler>
```

The following illustration compares default and specified compensation.
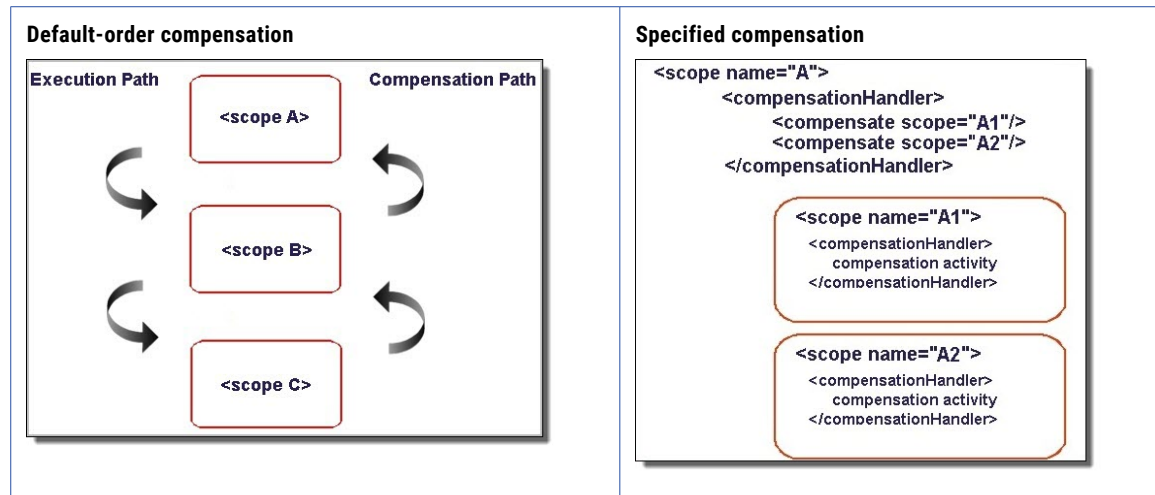
Implicit Compensation

In the case where a scope does not define a compensation handler, the execution engine provides an implicit compensation handler. This handler contains a single `<compensate>` activity at compensates all of the enclosed scopes that have completed and are eligible for compensation. In general, you should only provide a compensation handler when you have some application-specific logic to execute in order to undo some previously executed logic.

Default-Order Compensation Example

Default compensation processing is available from the compensate activity with the form `<compensate/>`. The activity invokes the compensation handlers on all of the enclosed scopes that are eligible for compensation. The scope's compensation handlers are invoked in the reverse order of completion. If a scope executed multiple times in a `<while>` loop, it is eligible for each execution instance where it completed without having a fault generated.

The following example shows how a confirmed purchase order is canceled. The `CancelPurchase` operation compensates the `SyncPurchase` operation in an already completed Invoke activity.

```
<scope>
    <compensationHandler>
      <invoke partnerLink="Seller" portType="SP:Purchasing"
        operation="CancelPurchase"
        inputVariable="getResponse"
        outputVariable="getConfirmation">
        <correlations>
          <correlation set="PurchaseOrder" pattern="request"/>
        </correlations>
      </invoke>
    </compensationHandler>
      <invoke partnerLink="Seller" portType="Sell:Purchasing"
        operation="SyncPurchase"
        inputVariable="sendPO"
        outputVariable="getResponse">
        <correlations>
          <correlation set="PurchaseOrder" initiate="yes"
              pattern="request"/>
        </correlations>
      </invoke>
</scope>
```

Specified Compensation Example

The following example shows how a Compensate activity is incorporated into a fault handler. For more information, see *Fault Handling*.

The following example shows how a Compensate activity is incorporated into a fault handler.

```
<faultHandlers>
  <catch faultName="lns:loanProcessFault"
    faultVariable="error">
    <sequence name="fault-sequence">
      <compensate scope="assessor-scope"/>
    ...
    </sequence>
  </catch>
</faultHandlers>
```

Adding a Compensation Handler to a Scope

A scope's activities can be compensated, or reversed when the scope is completed and when another activity causes compensation to begin.

To add a compensation handler to a scope:

1. From the Process Editor canvas, select a scope.

2. Drag a Compensation catch event into the scope or near the border of a collapsed scope, as the illustration shows.
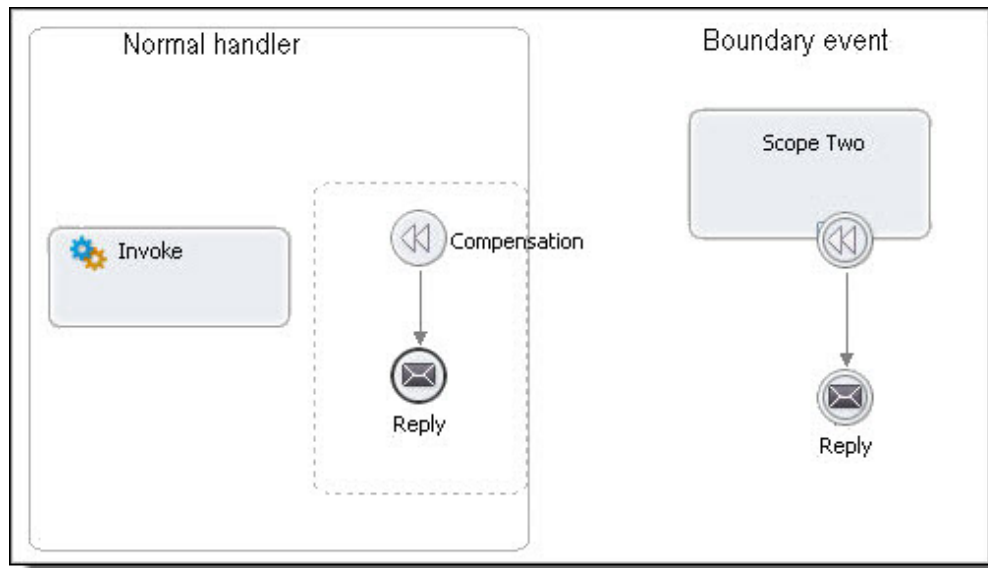
3. Drag activities into the compensation handler that will reverse the work of the scope's main activity. For a boundary event, you must link the handler to an activity to execute. The activity must be a downstream activity that has not yet executed.

4. From another scope, create an activity that triggers the compensation handler for the completed scope.

The following illustration shows a parent scope, children scopes, and their compensation handlers.
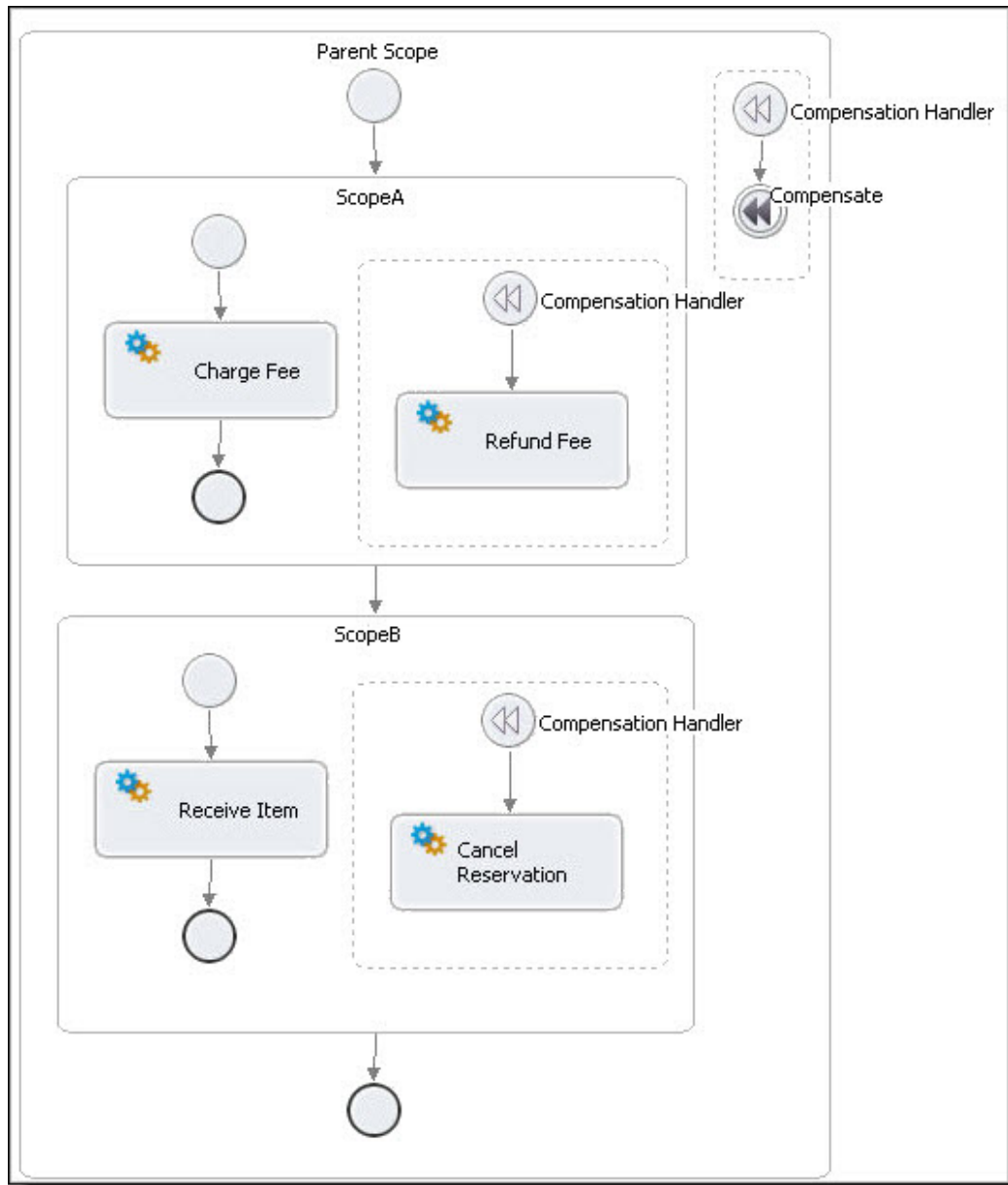
Compensating an Invoke Activity

The WS-BPEL 2.0 specification provides a shorthand construct for adding an optional compensation handler and fault handler to an invoke. You can add these handlers as boundary events.

CHAPTER 14

Correlation

CHAPTER 15

What is Correlation

Correlation is a construct for keeping track of a group of messages that belong together in one particular business partner interaction. Correlation matches messages and interactions with the business process instances they are intended for.

When a BPEL engine receives a message, it determines if it should create either a process instance or match an already-running process. The data within a correlation set is the signature that lets the engine match the message to the process expecting that message. You can add correlation to receive, pick, reply, invoke, and coordinate event handler activities for this purpose.

To use correlation, consider the following:

- Determine what activities need to be correlated. These activities should share one or more pieces of common data.
- Define one property that identifies the piece of common data.
- Define property aliases, a different one for each piece of common data in each activity that needs to be correlated.
- Create a correlation set that contains the property and property aliases.
- Add the correlation set to the activities being correlated.

As an example, for each correlation set, create a data property, such as `customerId`. For each message being correlated, create a property alias for a message part, such as a Receive's message part named `CustID`, an Invoke's input message part named `customerNum`, and so on.

Alternately, you can use an engine-managed correlation policy assertion, described in *Engine-Managed Correlation*.

The steps for adding correlation to a process are shown in the following illustration.

❶ Create Message Properties and Property Aliases

❷ Create a Correlation Set

❸ Initiate Correlation in an Activity

❹ Correlate Messages for a Process
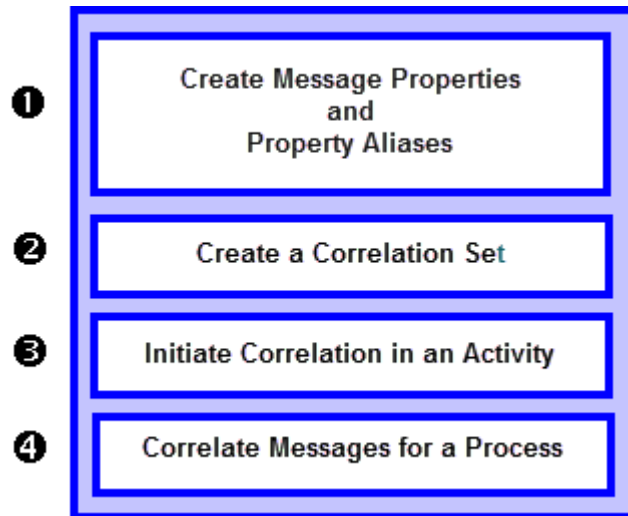
What is a Correlation Set

Select properties defined in a WSDL to use in a correlation set.

A *correlation set* is a set of properties shared by messages. A correlated message data that is the same as that in all other messages in the set. Since the element names of each message can be different, you need a mechanism to match up the same piece of data among the messages.

How does the mechanism work? For each correlation set, you create a data property, such as customerID. For each message being correlated, you create a property alias for a message part, such as a Receive's message part named custID, an invoke's input message part named CustomerNumber, and so on.

Properties

A message property to be used for message correlation must be a schema simple type like an `xsd:int` or an `xsd:string`.

Property Aliases

Properties can exist in WSDL messages, schema elements, and schema types. The property alias specifies the message part or query.

For example, a purchase ordering process might have an `OrderId`. property. Messages can store the value of `OrderId` in different parts with different names. A property alias identifies the part. A specific process instance is identified by a unique `OrderId` such that no two processes will ever have the same value for `OrderId`. This ensures that when a message arrives, it is dispatched to the correct process instance.

The following illustration shows how a message property alias is mapped to a property name. The property name is mapped to a correlation set. You can create as many correlation sets as you need.

Correlation Set	Property	Property Alias
CS 1	OrderId	CustomerId OrderRequest OrderResponse
CS 2	InvoiceId	POOrder Invoice CustomerID
	ShipNoticeId	OrderRequest OrderResponse

For more information, see:

- *Creating Message Properties and Property Aliases*
- *WSDL Syntax and Example for Property Names and Aliases*
- *Global and Local Correlation Sets*

WSDL Syntax and Example for Property Names and Aliases

Use a Process Developer wizard to automatically create variable properties and property aliases. (This is described in *Adding Variable Properties and Property Aliases*). The wizard generates WSDL code, as the following examples show.

The WSDL syntax for a property name is:

```
<wsdl:definitions name="NCName"
    xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop">
 <vprop:property name="NCName"
                 type="QName"?
                 element="QName"?/>
 ...
</wsdl:definitions>
```

The WSDL syntax for a property alias is:

```
<wsdl:definitions name="NCName"
    xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop">
 <vprop:propertyAlias propertyName="QName"
     messageType="QName"?
     part="NCName"?
     type="QName"?
     element="QName"?>
   <vprop:query queryLanguage="anyURI"?>?
       queryContent
   </vprop:query>
 </vprop:propertyAlias>
 ...
</wsdl:definitions>
```

Property and Property Alias Example

Consider the following message definition:

```
<wsdl:definitions name="messages"
      targetNamespace="http://example.com/taxMessages.wsdl"
      xmlns:txtyp="http://example.com/taxTypes.xsd"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
   <!-- define a WSDL application message -->
   <wsdl:message name="taxpayerInfoMsg">
      <wsdl:part name="identification"
          element="txtyp:taxPayerInfoElem" />
   </wsdl:message>
    ...
</wsdl:definitions>
```

The following WSDL fragment shows the definition of a property and its location in a particular field of the message:

```
<wsdl:definitions name="properties"
      targetNamespace="http://example.com/properties.wsdl"
      xmlns:tns="http://example.com/properties.wsdl"
      xmlns:txtyp="http://example.com/taxTypes.xsd"
      xmlns:txmsg="http://example.com/taxMessages.wsdl" ...>
   <!-- define a correlation property -->
   <vprop:property name="taxpayerNumber" type="txtyp:SSN" />
   ...
   <vprop:propertyAlias propertyName="tns:taxpayerNumber"
          messageType="txmsg:taxpayerInfoMsg"
          part="identification">
      <vprop:query>txtyp:socialsecnumber</vprop:query>
   </vprop:propertyAlias>
   <vprop:propertyAlias propertyName="tns:taxpayerNumber"
          element="txtyp:taxPayerInfoElem">
      <vprop:query>txtyp:socialsecnumber</vprop:query>
   </vprop:propertyAlias>
</wsdl:definitions>
```

The first `<vprop:propertyAlias>` defines a named property `tns:taxpayerNumber` as an alias for a location in the identification part of the message type `txmsg:taxpayerInfoMsg`.

The second `<vprop:propertyAlias>` provides a second definition for the same named property `tns:taxpayerNumber` but this time as an alias for a location inside of the element `txtyp:taxPayerInfoElem`.

The presence of both aliases means that it is possible to retrieve the social security number from both a variable holding a message of messageType `txmsg:taxpayerInfo` as well as an element defined using `txtyp:taxPayerInfoElem`.

Global and Local Correlation Sets

You can declare a correlation set for the process as a whole or for a scope. If you declare the correlation set locally for a scope, it is applicable only to the activities in the scope. You can hide a global correlation set by declaring a correlation set of the identical name in a scope.

CHAPTER 17

Creating Message Properties and Property Aliases

Create a property definition, and add it to an existing WSDL. Create a new WSDL, if desired, for a property based on a schema simple type. Select properties defined in a WSDL to use in a correlation set.

If your WSDL file does not include message property definitions, you can add them automatically to an existing WSDL or create a new WSDL for them. The property is automatically added to the corresponding process variables.

You use message properties and property aliases to create a correlation set. For an overview, see *What is Correlation?* and *What is a Correlation Set?*

In preparation for creating message properties and property aliases, consider the following:

- Which activities need to be correlated?
- Do these activities contain messages with a piece of common data?

If your WSDL file does not include message property definitions, you can add them automatically to an existing WSDL or create a new WSDL for them. The property is automatically added to the corresponding process variables.

Before you create message properties, ensure that your BPEL process references a WSDL file in the Imports section of the Outline view. See *Importing WSDL, Schema, and Other Resources* for details.

To add message properties and aliases, see:

- *Creating a Property Definition*
- *Creating a Property Alias*

Creating a Property Definition

Create a property definition, and add it to an existing WSDL. Create a new WSDL, if desired, for a property based on a schema simple type. You can create a property as follows:

1. From the Outline view, right-mouse click on Correlation Sets, and select **Add Correlation Set**.
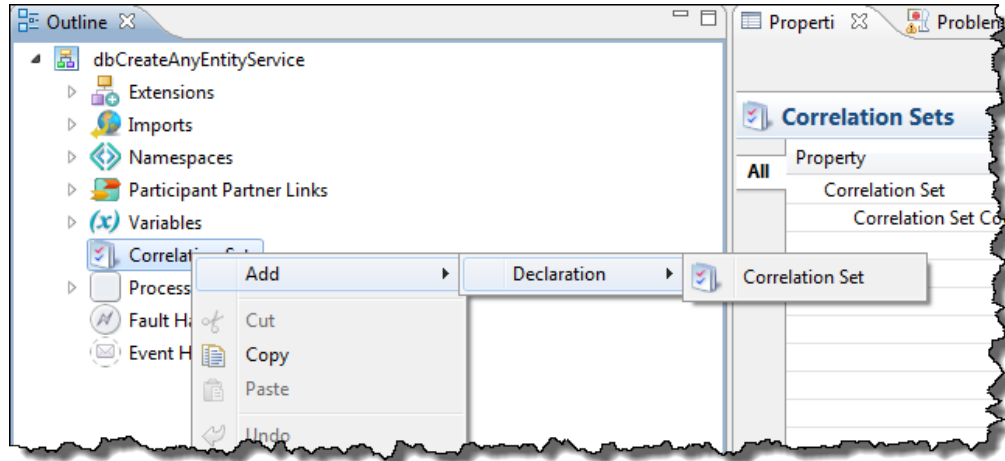
2. In the Correlation Set Properties dialog, select **New**.
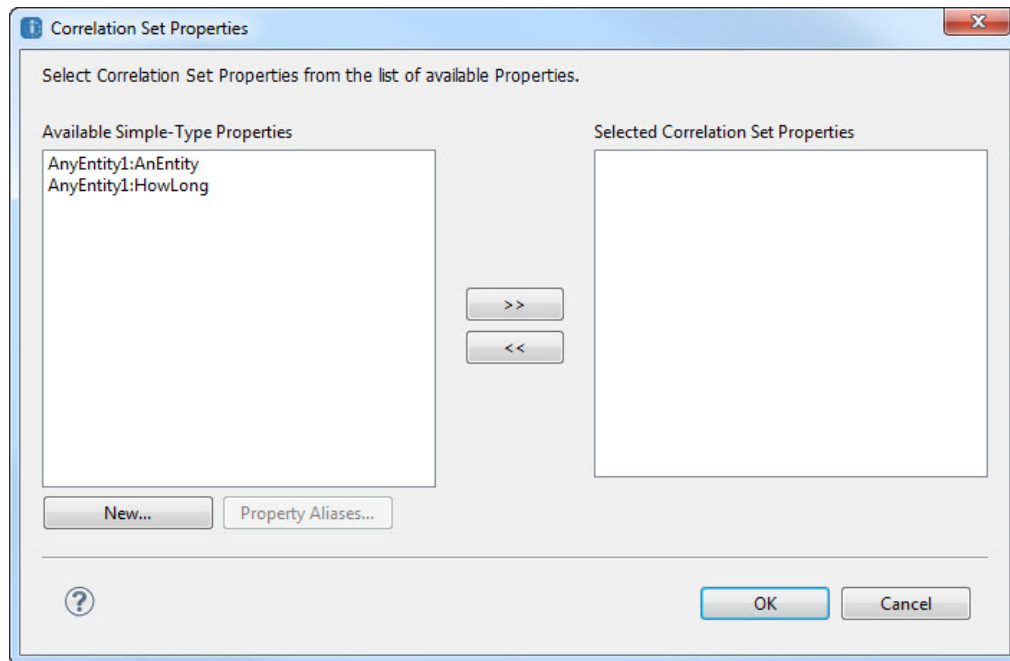
3. In the Property Definition dialog, select a Name for the property (for example, `OrderId`).
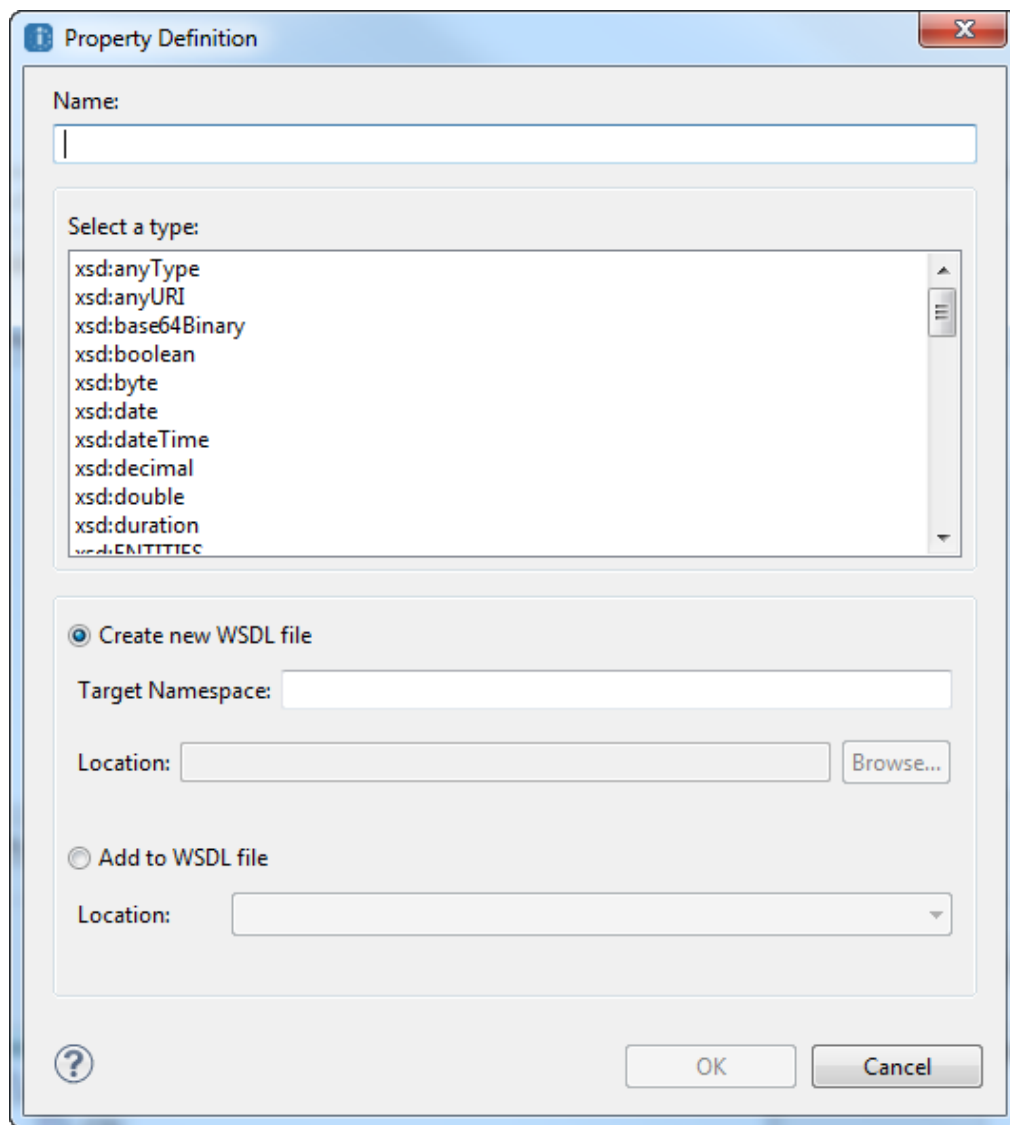
4. Select the property type: **schema type** or **schema element**.
   The property must be a schema simple type.

5. Select whether to add the property definition to an existing WSDL or to create a new WSDL. If the WSDL referenced in your process is a URL or otherwise shared by many partners, it is best to create a new WSDL.

6. To create a new WSDL, do the following:

   a. Type in a Target Namespace for the WSDL (for example urn:MyNewNamespace).

   b. Select **Browse**, and in the **Open** dialog, browse to a Project Explorer project or file system location.

   c. Type in a filename for the WSDL file, and click **Open**.

   d. Click OK and skip to Step 8.

7. To add to an existing WSDL, select a WSDL from the picklist. The list consists of WSDLs that were added to your process. Click **OK**.

8. In the Available Properties list of the **Correlation Set Properties** dialog, notice that the new property is listed.

9. Do one of the following:

    - Continue to *Creating a Property Alias*.

    - Click **OK** to close the dialog. You can add property aliases later. They are required before you can add a message correlation set.

If you create a new WSDL, Process Developer does the following:

- Adds the WSDL to the Imports section of the Outline view.

- Adds the new namespace to the process.

- Adds an <import> element to the new WSDL to reference the process' WSDL file. This allows you to select appropriate messages, types or elements for property aliases.

- Adds the new WSDL to Project Explorer, Participants, and Interfaces views.

Creating a Property Alias

Add, edit or delete a property alias for the selected property. Select the message part for the property alias. For a complex type, include the query.

1. Select a property from the Available Properties list in the **Add Correlation Set** dialog.

2. Select **Property Aliases**.

3. In the **Property Aliases** dialog, select **New**.

4. In the **Define Property Alias Details** dialog, select Dialog Button (...) next to the *Type* field to open the **Definition** dialog. The dialog contains messages, schema types and elements from namespaces imported into the process, as shown in the example.
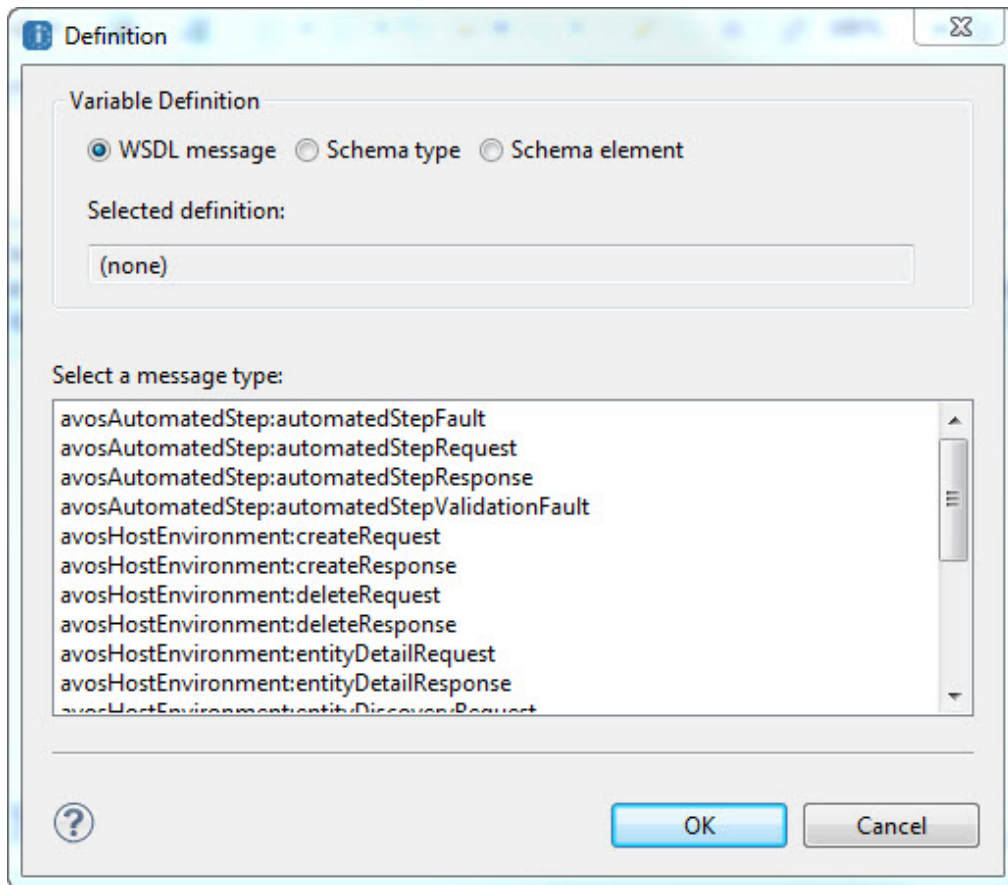
5. Select the variable definition: *WSDL Message*, *Schema type*, or *Schema element*. The list of available types appears, depending on the definition type you select.

6. Select an item in the list and click **OK**.

7. On the **Define Property Alias Details** dialog, select a variable *Part* for the property alias, as shown in the example.
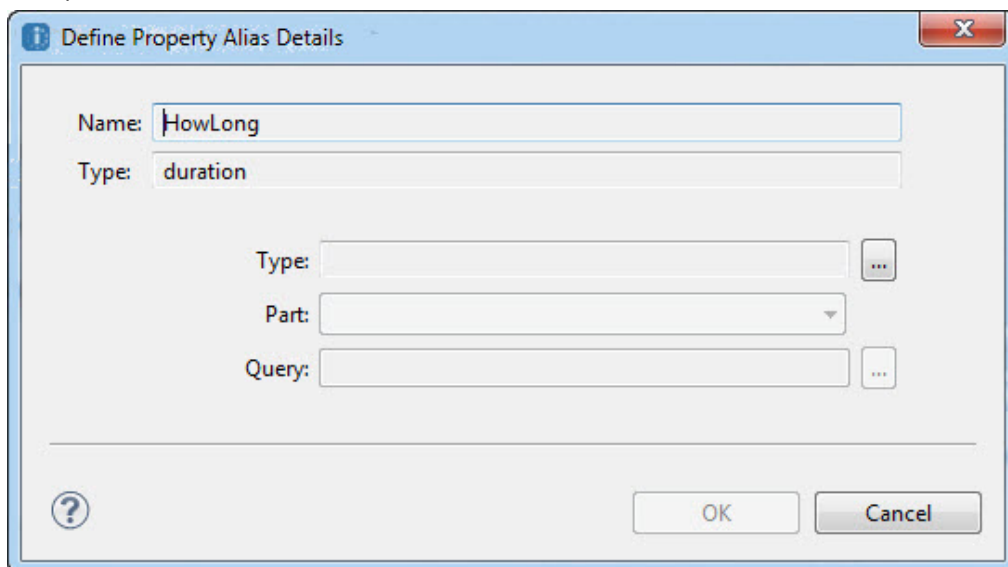
8. If the part is a complex type, type in a *Query* or select the Dialog button (**…**) to open the Query Builder and add the selection node from the part.

9. Click **OK**.

10. Repeat these steps to add property aliases for all messages, schema types, and schema elements that are included for the property. Your selections can depend on how you want to group messages for correlation.

After you add a property alias, Process Developer displays the property for the process variable, as the example shows.
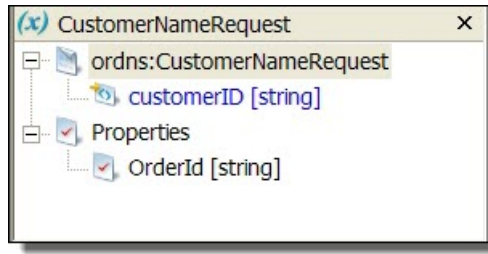
Process Developer also adds the property and property alias to the WSDL folder of your project.

Tips for creating property aliases:

- You can add property aliases one at a time.
- You can edit and remove property aliases as needed. The WSDL file and relevant process variables get updated correctly.

CHAPTER 18

Adding a Correlation Set

Add a correlation set to an activity.

Before adding correlation to an activity, you must create the correlation set.

To create the correlation set, you identify message, element, or schema properties from a WDSL file that map to correlation properties, as described below.

You can add a correlation set to process activities or to a scope. For more information, see *Global and Local Correlation Sets*.

To add a correlation set:

1. Ensure that a WSDL file is available that contains defined properties and property aliases. For more information, see *What is a Correlation Set?*

2. Do one of the following:

   - To add a correlation set for the process, from the Outline view, right-mouse click on Correlation Sets, and select **Add > Declaration > Correlation Set**

   - To add a correlation set to a scope, right-mouse click on the scope, and select **Add > Declaration > Correlation Set**

3. Select a property from the *Available Properties* list and move it to the *Selected Correlation Set Properties* list.
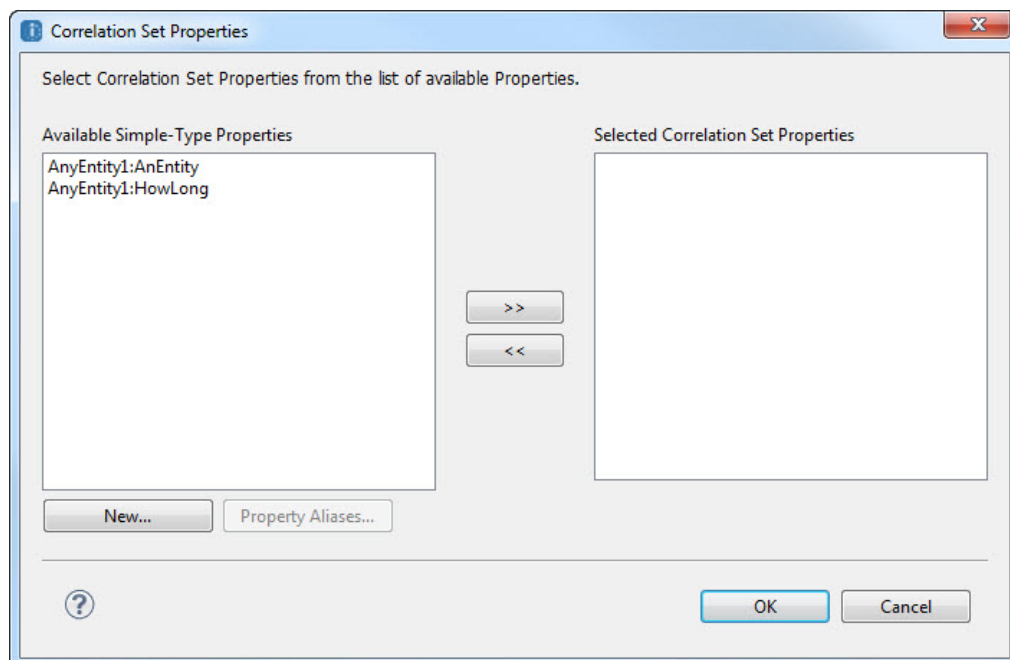
4. If you need more properties, move them to the right list as well.

Correlation Sets: Required and Optional Properties

The following table shows the required and optional properties for a correlation set.

Required Properties	Optional Properties
Set Name	Documentation.
Properties.	Comment.
	Extension Attributes and Extension Elements.

XML Syntax

Here is the XML syntax for a correlation set:

```
<correlationSets>?
  <correlationSet name="NCName" properties="QName-list"/>+
</correlationSets>
```

Example

Here is an example declaration for a correlation set:

```
<correlationSet name="PurchaseOrder"
    properties="cor:customerID cor:orderNumber"/>
```

XML Syntax

Here is the XML syntax for a correlation set:

```
<correlationSets>?
                              <correlationSet name="NCName" properties="QName-
list"/>+
                    </correlationSets>
```

Example

Here is an example declaration for a correlation set:

```
<correlationSet name="PurchaseOrder"
    properties="cor:customerID cor:orderNumber"/>
```

Correlation Sets: Required and Optional Properties

The following table shows the required and optional properties for a correlation set:

Required Properties	Optional Properties
Set Name	Documentation. See *Adding Documentation to a Process*.
Properties. See *Creating Message Properties and Property Aliases*.	Comment. See *Adding Comments to a Process*.
	Extension Attributes and Extension Elements. See *Declaring Extension Elements* and *Attributes*.

CHAPTER 19

Deleting a Correlation Set

Delete correlation sets from this BPEL process.

You can delete a correlation set from the process or from a scope.

To delete a correlation set from the process:

1. Click on a blank spot in the Process Editor canvas.

2. Select **Process** from the Process Developer main menu.

3. Select **Delete > Correlation Set**.

4. Select a correlation set and click **OK**.

To delete a correlation set from a scope:

1. In Outline view, expand a Scope node to view correlation sets.

2. Right-mouse-click on a correlation set, and select **Delete**.

CHAPTER 20

Adding Correlations to an Activity

For a discussion of using engine-managed correlation versus correlation sets, see *Correlation Sets and Engine-Managed Correlation*.

You can add correlations to the following kinds of Web service interaction activities:

- Receive
- Pick (onMessage branch)
- Event handler (onEvent variant)
- Invoke
- Reply

After you add correlation to an activity and save your process file, the Problems view shows warning messages for the related activities that should be correlated.

For a discussion of correlation attributes that you must set, see *Initiating and Setting Patterns for Correlation*.

To add correlation, see *Adding Correlation to a Receive, OnMessage, OnEvent, or Reply* and *Adding Correlation to an Invoke Activity*.

CHAPTER 21

Rules for Declaring and Using Correlation Sets

Here are some rules for declaring and using correlation sets:

- A correlation set is declared within a scope, similar to a variable declaration. You can declare both global and local correlation sets.

- You can hide a correlation set in an outer scope by declaring a correlation set with an identical name in an inner scope. If you declare the correlation set locally for a scope, it is applicable only to the activities in the scope.

- A correlation set can be initiated only once during the lifetime of the scope it belongs to.

- A correlation set must be initiated.

- After a correlation set is initiated, the values of the properties for a correlation set must be identical for all messages in all operations in the scope.

Correlation Sets and Engine-Managed Correlation

To correlate inbound messages, you have two choices. You can:

- Create and use correlation sets or you can allow the Process Developer engine to automatically manage message correlation.

- Direct the Process Developer engine to automatically correlate inbound messages targeted for a receive, onMessage, or onEvent by adding a policy to the My Role partner link in the Process Deployment Descriptor file.

For details, see Engine-Managed Correlation.

CHAPTER 23

Event Handling

This chapter includes the following topics:

What is Event Handling

Event handling is an activity that runs concurrently with a scope. Events can be one of two types: a message event or an alarm. When a message event or alarm occurs, the event handler associated with it invokes an activity.

Event handlers are especially helpful for events and requests that cannot be scheduled relative to the main activity, but may occur at unpredictable times. For example, a customer can cancel an order that is being processed.

Event handlers are considered part of the normal behavior of the scope, unlike fault and compensation handlers that take over normal processing.

An event handler can be attached to the process as a whole or to a scope. One or more events can occur and be handled at any time while the corresponding scope is active.

An event handler cannot be enabled until a process instance is created, meaning it cannot create the process instance itself.

Event Types

An event can be an incoming message that corresponds to a request/response or one-way operation in a WSDL. For example, a status query is likely to be a request/response operation, whereas a cancellation can be a one-way operation.

An event can also be an alarm that goes off after a set time or lasts for a specified time, and optionally repeats.

Examples of Event Handler Activities

When a message that triggers an event is received, one of the following can be the likely response by an event handler:

- Send a reply.

- Terminate the process instance. For example, an order is cancelled, so the process instance should be cancelled, and there is no ongoing work to be undone and compensated.

- Throw a fault to cause the ongoing work to be undone and compensated.

Adding Event Handlers

You can add event handlers to the process as a whole or to a scope. You can add two types of event handlers:

- onEvent Event Handler

- an onAlarm Event Handler

For conceptual information, see *What is Event Handling?*

XML Syntax

```
<eventHandlers>?
  <!-- there must be at least one onEvent
       or onAlarm handler -->
  <onEvent partnerLink="NCName" portType="QName"?
    operation="NCName"
    (messageType="QName" | element="QName")?
    variable="BPELVariableName"?
     messageExchange="NCName"?>*
  <correlations>?
    <correlation set="NCName" initiate="yes|join|no"/>+
  </correlations>
  <fromParts>?
    <fromPart part="NCName" toVariable="BPELVariableName"/>
  </fromParts>
   <scope...>...</scope>
  </onEvent>
  <onAlarm>*
   (
     <for expressionLanguage="anyURI"?>duration-expr</for>
     |
     <until expressionLanguage="anyURI"?>deadline-expr</until>
   )
    <repeatEvery expressionLanguage="anyURI"?>?
     duration-expr
    </repeatEvery>
  <scope...>...</scope>
  </onAlarm>
</eventHandlers>
```

Adding an onEvent Event Handler

The `onEvent` element indicates that the event specified waits for a message to arrive. This element is very similar to a receive activity, except that a message event cannot create a new process instance. Also, an `onEvent` must contain a scope activity and optionally uses a variable local to the event handler that contains the message received from a partner.

An incoming message can correspond to a request/response or one-way operation in a WSDL. When the operation is a request/response type, the event handler is expected to use a reply to send back the response.

Required Properties	Optional Properties
Participant	Variable
Operation	Port Type
	Correlations.
	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images.
	Execution State
	Message Exchange.
	Extension Attributes and Extension Elements.

To add an onEvent event handler for the process:

Ensure that you have added a participant for the event handler before you can complete the properties for it.

1. Click on the Event Handlers tab of the Process Editor.
2. Drag a **Message catch event** to the Event Handlers canvas.
3. From the Properties view, select a participant, operation, and optionally select a Variable Definition and type in a Variable name. This variable is an implicit scope variable that can be used by other activities.
4. Drag an activity into the `onEvent`'s scope to respond to the event, such as a reply or exit.
5. Fill in the properties for the activity that handles the event.

The following illustration shows an example of an onEvent event handler added for the process. If you collapse the event handler, you can add a background color in the Properties view.
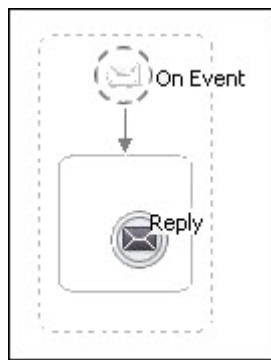
To add an onEvent event handler for a scope:

1. Ensure that you have added a WSDL file to Project Explorer containing the participant information needed.
2. Display a BPEL file in the Process Editor, and select a scope.

3. Drag a **Message catch event** to the scope.

4. Fill in the properties of the `onEvent`, including the participant, operation and optionally type in a variable. This variable is an implicit scope variable that can be used by other activities.

5. Drag an activity into the `onEvent`'s scope to respond to the event, such as a reply or exit.

6. Fill in the properties for the activity that handles the event.

The following illustration shows an example of an `onEvent` and an `onAlarm` event handler added for a scope. The activity responding to each event is a reply.
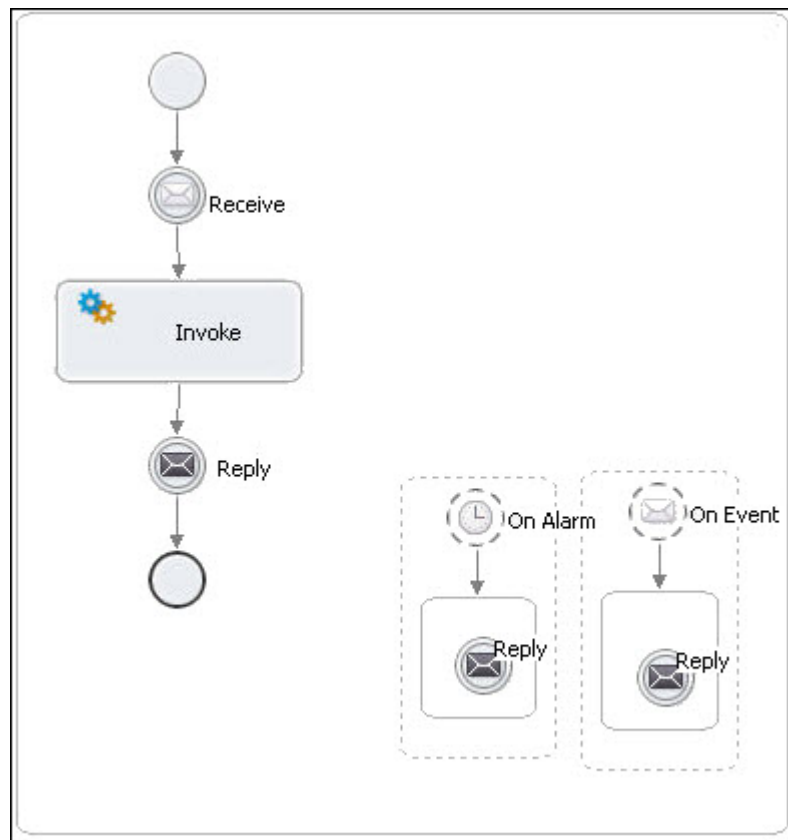
Adding an onAlarm Event Handler

The `onAlarm` element defines a timeout event, using either the *duration (for)* or *deadline (until)* attribute. The *duration* attribute specifies the amount of time after which the event is signaled. The clock for the duration starts when the associated scope starts. The alternative *deadline* attribute sets the specific point in time when the alarm is fired. Optionally a *repeat every* attribute can be set to fire an alarm repeatedly each time the interval period expires. The *repeat every* attribute can be used alone or with an *deadline* or *duration* attribute.

For example, you might set an alarm to go off in five minutes and then repeat the alarm every 30 seconds.

An alarm time can be set for the process using the data provided within the message that creates a process instance, as shown in the following example.

In the WSDL file, a complex message is defined with a part specifying *processDuration*.

```
<wsdl:definitions
  targetNamespace="http://www.example.com/wsdl/exmple"
```

```
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     ...>
      <wsdl:message name="orderDetails">
       <part name="processDuration"
           type="xsd:duration"/>
      </wsdl:message>
   </wsdl:definitions>
```

The message type above is used in the `onAlarm` value in the BPEL process:

```
<process name="orderCar"
  xmlns:def="http://www.example.com/wsdl/example" ...>
  ...
  <eventHandlers>
    <onAlarm
      <for>$orderDetails.processDuration</for>
    </onAlarm>
  </eventHandlers>
  ...
  <variable name="orderDetails"
      messageType="def:orderDetails"/>
  </variable>
  ...
  <receive name="getOrder"
    partnerLink="buyer"
    operation="order"
    variable="orderDetails"
    createInstance="yes"/>
  ...
</process>
```

In the above example, the `onAlarm` element specifies a timer event that is fired when the duration specified in the *processDuration* field in the *orderDetails* variable is exceeded. The value of the field is provided via the *getOrder* activity that receives a message containing the order details and causes the creation of a process instance for that order.

Required Properties	Optional Properties
Alarm Expression	Repeat Every (can be used with Alarm Type)
Alarm Type or Repeat Every	Comment.
	Documentation.
	Setting Visual Properties and Using Your Own Library of Images.
	Execution State.
	Extension Attributes and Extension Elements.

To add an onAlarm event handler for the process:

1. Click on the Event Handlers tab of the Process Editor.

2. Drag a **Timer catch event** to the Event Handler canvas.

3. In the Properties view, optionally select the Dialog button at the end of the *Repeat Every* row. Type in or compose an expression for the interval. Use this attribute alone or in conjunction with a Wait Type.

4. Set the *Wait Type* as one of the following:

   a. Select *Deadline Expression* to type in or compose an expression representing the deadline (wait until) for the alarm.

b. *Duration Expression* to type in or compose an expression representing the duration (wait for) for the alarm.

c. *Duration* to select the number of days, hours, minutes, or seconds to wait.

d. T time in X days to set a deadline that is X days plus T hours. The expression resolves to an expression such as:
```
xsd:dateTime(current-date()) + xsd:dayTimeDuration("P3DT15H20S")
```

5. Drag an activity into the event handler to respond to the event, such as a reply or exit.

6. Fill in the properties for the activity that handles the event.

The following illustration shows an example of an `onAlarm` event handler added for the process.
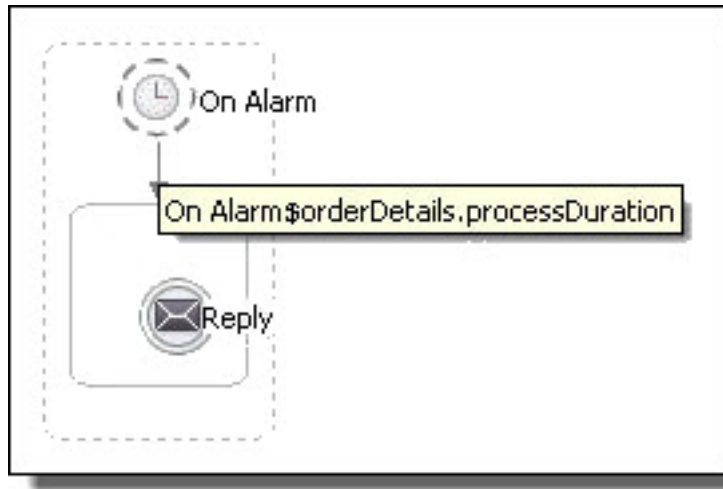
Tip: If you collapse the event handler, you can add a background color in the Properties view.

To add an onAlarm event handler for a scope:

1. Display a BPEL file in the Process Editor, and select a scope.

2. Drag a **Timer catch event** to the scope.

3. In the Properties view, fill in properties as described above for the process.

4. Drag an activity into the event handler to respond to the event, such as a reply or terminate.

5. Fill in the properties for the activity that handles the event.

The following illustration shows an example of an `onAlarm` and an `onEvent` event handler added for a scope. The activity responding to each event is a reply.
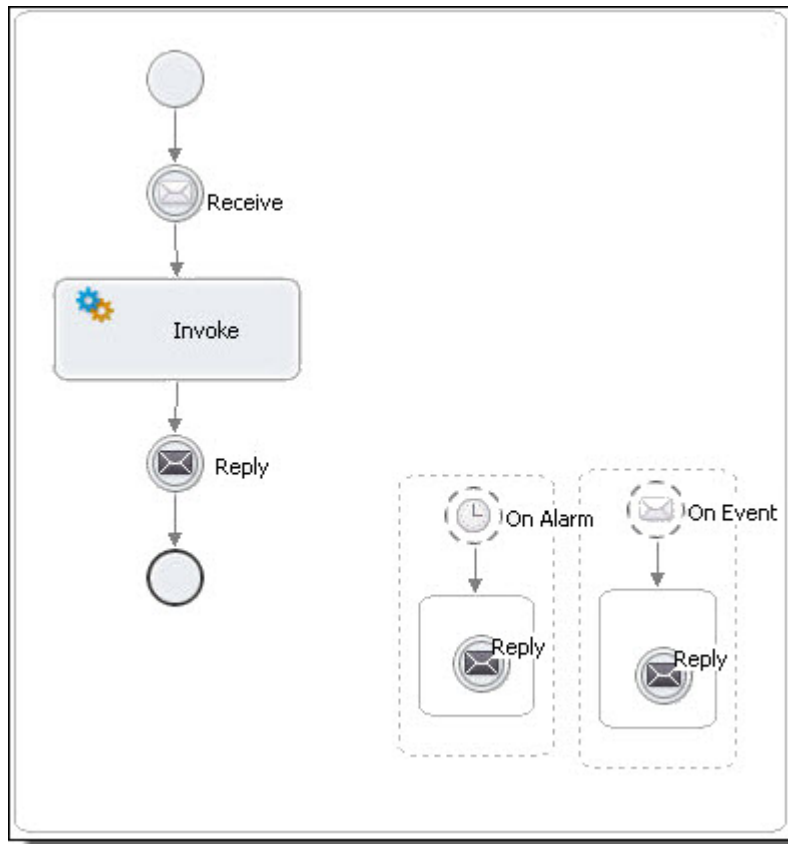
Processing Rules for Events

An alarm event occur only once, unless it is modified by a *repeat every* attribute.

A message event can occur multiple times. For example, while an order is being processed inside a scope, changes for the order can be accepted via a message event.

Multiple message and alarm events can occur concurrently. If your process contains two concurrent scopes with shared variables and event handlers, be sure to enable *isolated* for the scopes. Enabling this ensures consistent access to shared variables.

Adding Boundary Events

The Business Process Model and Notation (BPMN) 2.0 specification includes the concept of a boundary event. In Process Developer, a BPMN *boundary event* is a BPEL scope handler (event, fault, or compensation) that is dropped onto the boundary of a bordered activity. You can use a boundary event as a shortcut to creating a scope and its interior scope handler, providing a cleaner-appearing design. The event handling is shown at the same level as the main activity. Behind the scenes, Process Developer creates the BPEL code to wrap the activity in a scope for valid execution.

The following illustration shows a normal event handler compared to a boundary event on a scope.
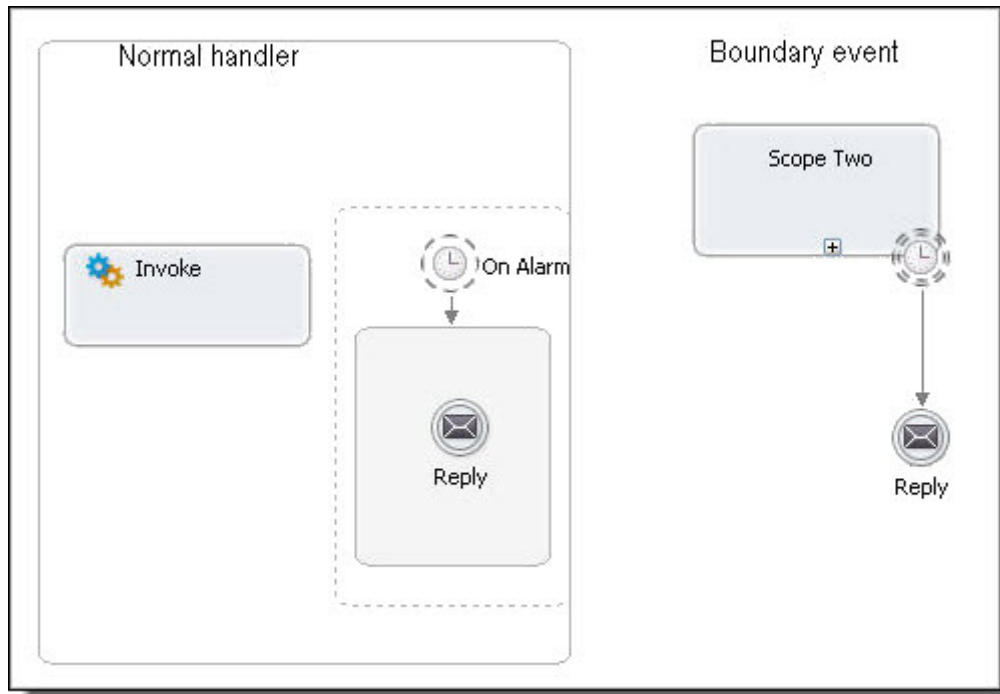
You can add a boundary event to any bordered activity. The following example shows an invoke with an `onEvent` handler that escalates a repair request from Standard to Premium.
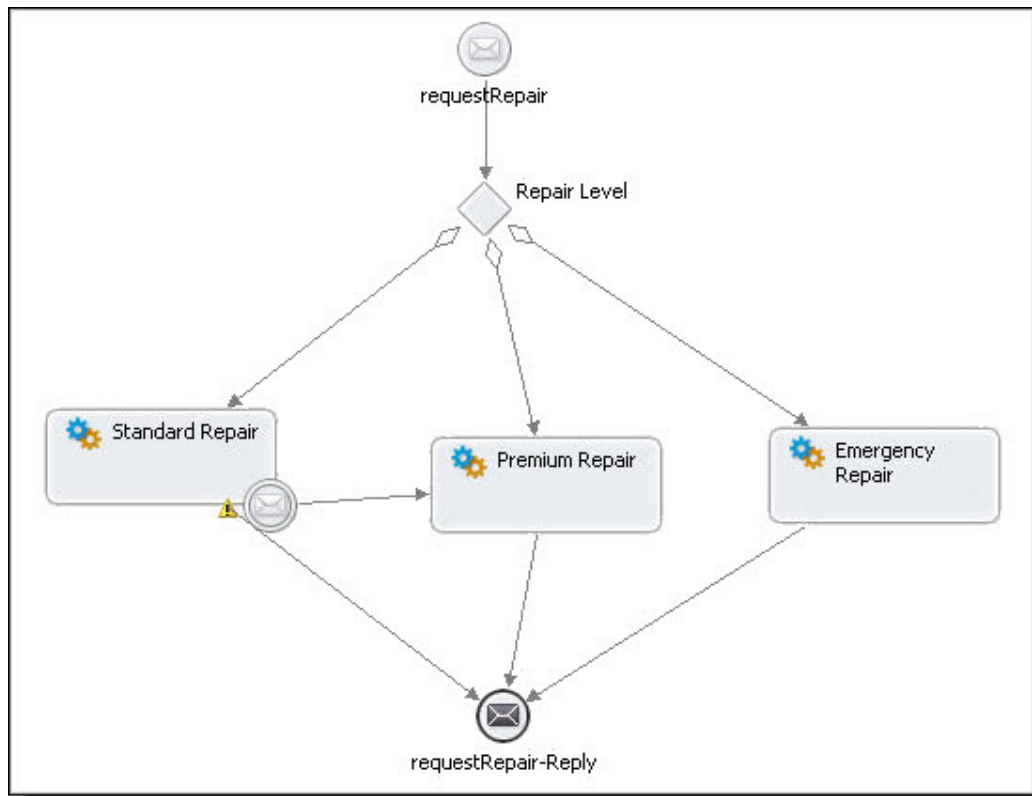
The properties of a boundary event are:

- Must be one of `onEvent`, `onAlarm`, `Catch`, `Catch All`, or `Compensation`.
- Must be dropped only onto the boundary (border) of any rounded border activity, such as a scope or invoke.
- Must use outbound links to activities that have not yet executed (downstream activities) or are not already dead path (branches of an if container).
- Outbound links cannot have conditions.
- Can be defined as interrupting or non-interrupting (discussed below).
- Interrupting boundary event is a Process Developer extension to BPEL.
- Only available in the BPMN editor, not the Classic editor.

Here are some examples of boundary events.

Non-interrupting onAlarm or onEvent

In the following illustration, the collapsed scope displays an `onAlarm` handler on its boundary. The handler can execute in parallel with the activities in the scope. A boundary event does not have a container like the `onAlarm` or `onEvent` handler that is dropped into a scope. Rather, the boundary event links to one or more activities to execute. The links cannot have conditions.
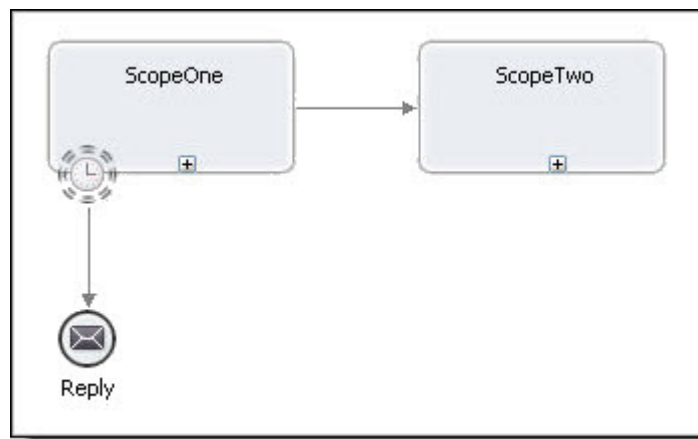
In the example above, the `onAlarm` boundary event behaves exactly like a scope `onAlarm` handler. The same processing rule applies: activities in the main scope execute in parallel with the `onAlarm` activity, if the alarm is triggered. This makes the boundary event non-interrupting. The linked activity (the reply in the above example) cannot be not part of the main process.

The Interrupting Boundary Event property is disabled by default, as the illustration shows:
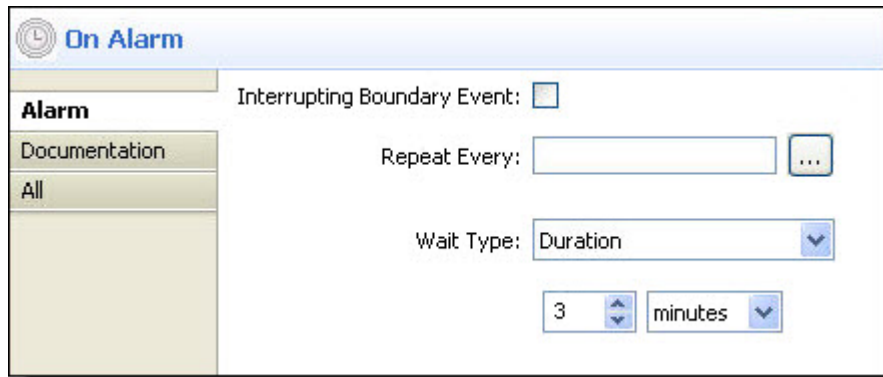
Interrupting onAlarm or onEvent

If desired, you can add a new processing rule to an `onAlarm` or `onEvent` by enabling the Interrupting property, shown above. In this case, the main scope terminates when the event is triggered.

Some use cases for an interrupting boundary event are as follows:

- `onEvent`: cancel an order
- `onAlarm`: request is no longer relevant, don't bother

An interrupting boundary event can be connected back to a downstream activity in the main process. Note that this behavior is an extension to WS-BPEL 2.0. Process Developer uses the Mutually Exclusive Transitions extension to execute an interrupting boundary event.

In the following illustration, the collapsed scope contains an interrupting `onAlarm` that, when triggered, terminates the main scope. Only one path is executed.
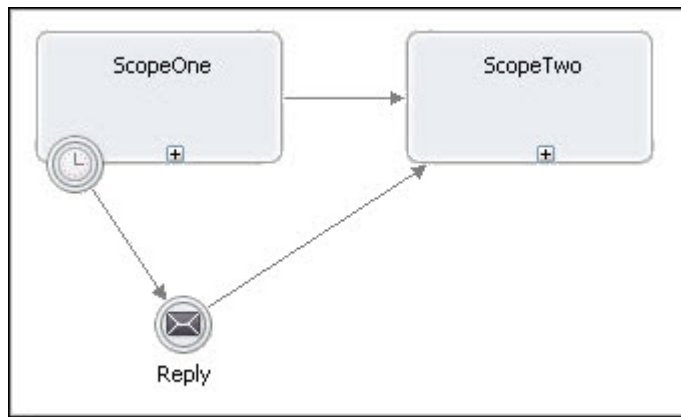
Note the difference in icon appearance for non-interrupting and interrupting boundary events.
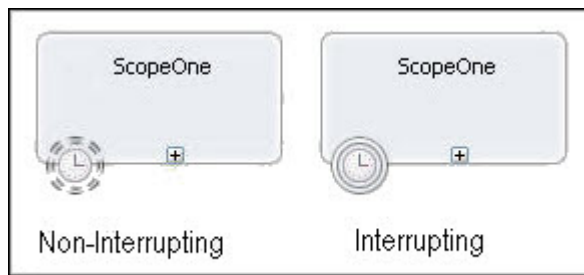
You can also add a boundary event for a fault or compensation handler. There are some minor exceptions as described in *Catch and Catch All Boundary Events and Compensate, Compensate Scope and Rethrow*.

See also:

- Adding a Fault Handler for a Scope
- Adding a Fault Handler as a Boundary Event for an Invoke Activity
- Adding a Compensation Handler to a Scope
- Compensating an Invoke Activity
- Using a Variable from a Catch or Interrupting OnEvent Boundary Event

Catch and Catch All Boundary Events and Compensate Compensate Scope and Rethrow

There are some BPEL activities that cannot be used as the target of a catch or catch all boundary event.

You cannot add a compensate, compensate scope, or rethrow activity as a linked target of a catch or catch all. The catch and catch all events are interrupting events, triggered by the termination of the scope they are attached to. As such, they get executed outside of the scope. The work of the compensate, compensate scope, and rethrow activities is to execute within a scope. These activities must be added inside a scope's catch or catch all handler.

Using a Variable from a Catch or Interrupting OnEvent Boundary Event

An interrupting `onEvent` boundary event as well as a catch boundary event are Process Developer extensions. As such, the variable handling for these events differs from normal BPEL fault and event handling.

Catch Boundary Event Variable Handling

If you want to define a variable for a catch boundary event, you must select an existing process variable or create a new process variable in the Fault Variable picklist.

This differs from defining a variable in a normal scope catch handler, where you can type in a special fault variable name.

This difference arises because in a normal catch handler, the fault variable is used within an activity that is enclosed by the scope. In an interrupting boundary event, there is no activity within the event handler. Instead, the boundary event links to an activity in the process or another scope.

Behind the scenes, Process Developer generates the activities needed in order to use boundary event variables. Process Developer creates an implicit `parameters` variable for the catch that is available only within a hidden scope containing the catch boundary event. Therefore, to define a variable for use in another part of the process, such as a reply with fault, you must select a process variable or create a new process variable. Process Developer maps the implicit `parameters` variable to the process variable. See the example below.

Interrupting OnEvent Boundary Event

The variable handling is essentially the same as for a catch boundary event. In the Properties view of an interrupting `OnEvent` boundary event, you can select an existing process variable or create a new variable in the Variable drop-down.

Example

The following illustration shows the Properties for an interrupting `OnEvent` boundary event.
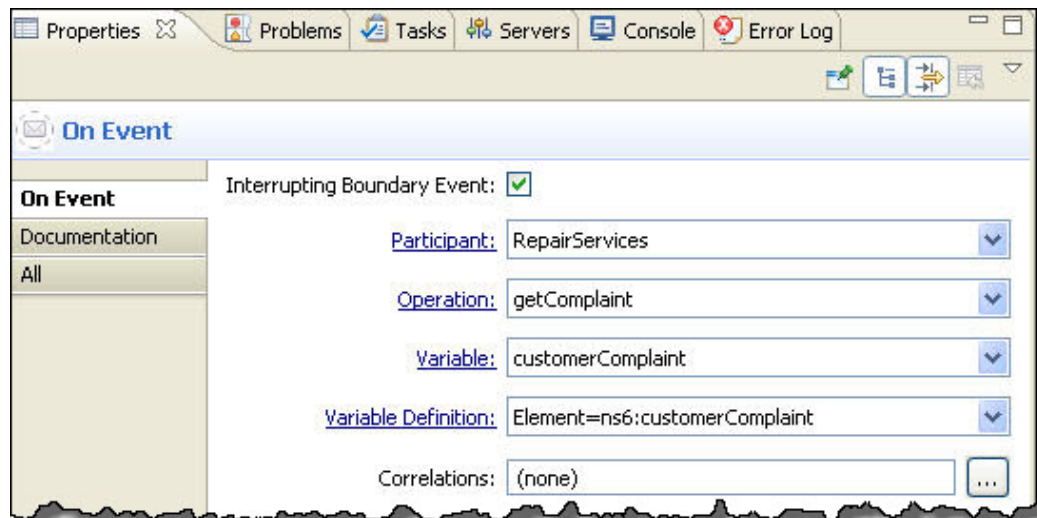
In the BPEL source code, notice that a scope with an assign is added. The assign copies the implicit scope variable, parameters, to the process variable, *customerComplaint*.

```
<bpel:eventHandlers>
            <bpel:onEvent aei:boundaryEvent="interrupting"
element="ns6:customerComplaint"
                            ext1:handleIMAInEnclosingScope="true"
                            operation="getComplaint"
                            partnerLink="RepairServices"
                            variable="parameters">
            <bpel:scope>
                <bpel:assign ext1:mutexTransitions="yes">
                    <bpel:sources>
                        <bpel:source linkName="L5"/>
                    </bpel:sources>
                    <bpel:copy>
                        <bpel:from variable="parameters"/>
                        <bpel:to variable="customerComplaint"/>
                    </bpel:copy>
                </bpel:assign>
            </bpel:scope>
            </bpel:onEvent>
</bpel:eventHandlers>
```

CHAPTER 24

Fault Handling

This chapter includes the following topics:

What is BPEL Fault Handling

Fault handling in a BPEL process refers to the procedures performed when an exception or unanticipated condition occurs during normal processing. The work performed restores state so that predictable results always occur. Fault handling differs from compensation handling in that fault handling happens when a fault occurs within a scope whereas compensation reverses the work of a successfully completed scope. This means that the fault handler has access to information contained within the scope.

A fault can occur for the following reasons:

- A web service operation cannot complete successfully, and the service returns a fault

- An internal process error occurs, and a standard BPEL fault is thrown. Refer to the list of *BPEL Standard Faults* in the *Process Developer Help* for more information.

- A `<throw>` or `<rethrow>` activity throws a fault

- A platform-specific fault, such as a communication failure, occurs in a BPEL process instance

Fault handling can be global or local: you can add fault handlers to the process as a whole or to a scope within the process.

When a fault occurs, normal processing is terminated, and control is transferred to the corresponding fault handler, as defined in the `<faultHandlers>` section of the process or scope.

Note that the process does not enable compensation for a scope in which a fault handler is invoked.

Fault handlers do not rely on state to determine which nested scopes have completed successfully.

For a discussion of fault handling and compensation handling for a scope, see *Lifecycle of a Scope*.

Defining Catch and CatchAll Fault Handlers

You can define two types of fault handlers:

- A `<catch>` fault handler that defines a set of custom fault-handling activities that execute based on an optional fault name and/or fault variable. If the fault name is missing, then the catch intercepts all faults with the same type of fault data.

- A `<catchAll>` fault handler that executes if a thrown fault is not caught by a `<catch>` fault handler. Catch all is a variant of catch. It does not specify a fault name or variable to catch.

You can define `<catch>` and `<catchAll>` handlers for the process on the Process Editor Fault Handlers tab, as described in *Adding a Fault Handler*.

The following illustration shows an example of fault handlers created on the Process Editor canvas.
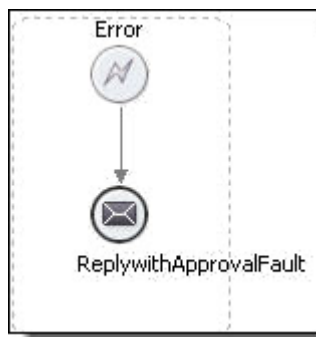
You can also define fault handlers for a scope, as the error catch event, labeled *Catch All*, shows in the following illustration.
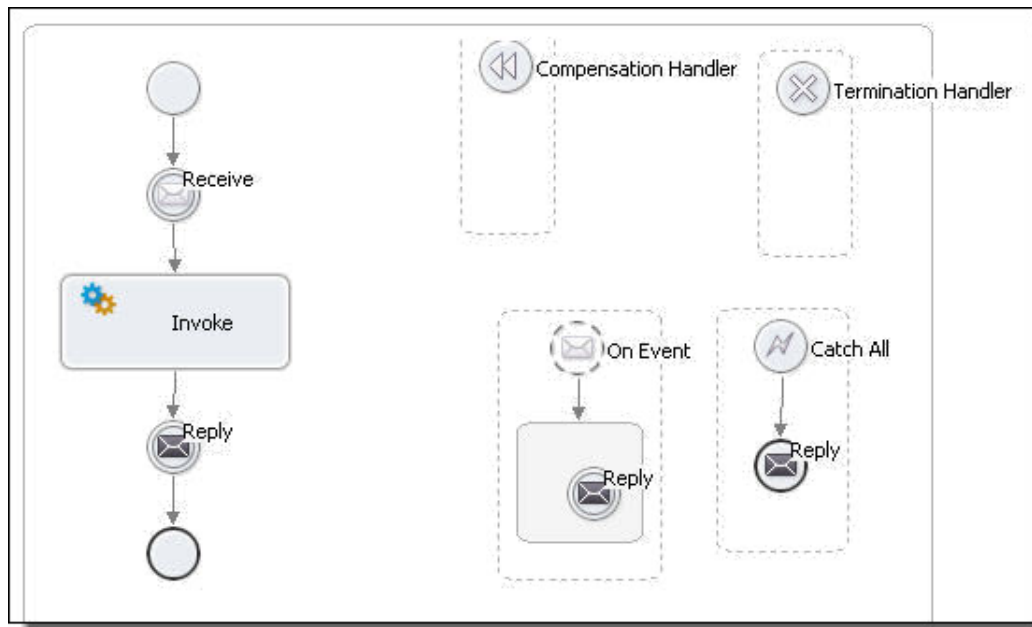
You can define each catch activity to intercept a specific kind of fault, defined by a globally unique fault name and a variable associated with the fault. Optionally, you can have a defined type associated with the fault variable, either a message type or an element type.

XML Syntax

```
<faultHandlers>?
  <catch faultName="QName"?
    faultVariable="BPELVariablename"?
    (faultMessageType="QName" | faultElement="QName")?>*
    activity
  </catch>
  <catchAll>?
    activity
  </catchAll>
</faultHandlers>
```

Required Properties	Optional Properties
Catch only: Fault name only or fault variable only or fault name plus fault variable	**Catch only:** Fault name, Fault variable definition, Fault variable
	Comment. See Adding Comments to a Process
	Documentation. See Adding Documentation to a Process
	See Setting Visual Properties and Using Your Own Library of Images.
	Execution State. See Viewing the Execution State of an Activity or Link
	Extension Attributes and Extension Elements. See Declaring Extension Elements and Attributes.

Catch Fault Name and Fault Variable

The fault name and fault variable/variable definition are optional attributes, but you must provide at least one of them for a `<catch>`. These attributes determine which catch in a fault handler executes when a fault is thrown.

If you do not define either a fault name or variable, the catch is executed as a `<catchAll>`.

In most cases, the values for these attributes are from the WSDL for a service being invoked. The name of the fault is typically the name of fault on an operation with the target namespace as the qualifying namespace value. The variable for a fault is typically the message sent when an operation in WSDL throws a fault. In the case where a catch includes a fault variable, that variable must be defined within the catch. This variable gets populated with the data from the fault when the catch executes. In addition, Process Developer provides special handling for undeclared faults.

For more details about the use of fault names and variables, see the following topics:

- Fault Handling Processing Rules
- Tips on Fault Handling
- Catching Undeclared and SOAP Faults

CatchAll Handler Examples

The `<catchAll>` catches any faults that were not caught by an existing catch block. The benefit of this construct is that you can be sure that your fault handler has an opportunity to execute your fault handling code in the event a fault is thrown. One drawback is that you will not have any details regarding the type of fault that was thrown since there is no fault name or fault variable available to a `<catchAll>`, per se. However, Process Developer provides custom functions to handle undeclared faults in a `<catchAll>`. This

construct is best used in cases where you do not care what fault is thrown or in cases where you want to be sure that no faults get past your existing fault handler logic.

The default `<catchAll>` fault handler is as follows:

```
<catchAll>
  <sequence>
    <compensate/>
    <rethrow/>
  </sequence>
</catchAll>
```

Fault Handling for Service Invocations

A fault response to an invoke activity uses the definition of the fault in the WSDL operation. A WSDL fault is defined by using the target namespace of the port type and the fault name.

Example: Purchase Order BPEL Process

In a normal purchase order BPEL process, the following steps occur:

1. A customer sends in a purchase order.

2. The BPEL process receives a purchase order and performs tasks for calculating shipping and production costs.

3. The BPEL process returns an invoice to the customer.

If something goes wrong, the process returns an error. Fault messages are defined in the WSDL file and used in the BPEL process as described in the following example:

WSDL Fault Definition	BPEL Fault Handling for Service Invocation

``
<portType name="POrderPT">
 <operation name="sendPOrder">
    <input message="..."/>
    <output message="..."/>
    <fault name="cannotCompletePO"
      message="pos:orderFaultType"/>
 </operation>
</portType>
``
	``
	<faultHandlers>
	 <catch
	   faultMessageType="ns1:errorMessage"
	   faultName=="lns:cannotCompletePO"
	   faultVariable="POFault">
	   <reply partnerLink="purchasing"
	   operation="sendPOrder"
	   variable="POFault"
	   faultName="lns:cannotCompletePO"/>
	 </catch>
	</faultHandlers>
	``

In the WSDL definition above, the operation called `sendPOrder` specifies the fault named `cannotCompletePO` that is used by the BPEL process if the purchase order cannot be completed. In the BPEL process fault handlers section, the catch activity contains a reply defined to return a message in the `POFault` variable in the event that a fault is received.

When an invoke operation returns a fault message, it causes a fault in the current scope. The fault variable in the corresponding `<catch>` is initialized with the fault message received.

If a Web service operation is not defined with a fault message, you can use the Process Developer custom functions to catch undeclared faults.

For more information, see *Adding a Fault Handler as a Boundary Event for an Invoke Activity* and *Catching Undeclared and SOAP Faults*.

Adding a Fault Handler

Select the variable definition, either message type or element. Then select a variable from the list.

Your BPEL process can catch standard faults as well as faults from a throw or re-throw activity or service invocation. Once caught, a fault is handled by a catch activity you specify, such as a reply containing an error message or a compensate activity. While a <catch> or <catchAll> can have only a single child activity, this activity can be a structured activity that can have its own children.

There are several ways to add fault handlers, including:

- Adding a Fault Handler for the Process
- Adding a Fault Handler for a Scope
- Adding a Fault Handler as a Boundary Event for an Invoke Activity

For background information, see *Fault Handling*.

Adding a Fault Handler for the Process

You can add multiple catch handlers and one catch-all handler for the process.

For explanations of these handlers, see Defining Catch and CatchAll Fault Handlers and Catching Undeclared and SOAP Faults.

To add a catch handler for the process:

1. Click on the Fault Handlers tab of the Process Editor canvas.
2. Drag an **Error catch event** to the canvas.
   A catch handler is added and is selected.
3. In the Properties view, select the following optional values:
   a. Fault Name. See Fault Handling Processing Rules.
   b. Fault Variable. Type in name for the variable described by the variable definition. For a discussion on whether or not to add a fault variable, see Rules for Catching Faults in a Catch Activity. If you have selected a Fault Name, the fault variables picklist displays only compatible variables.
   c. Variable Definition. Select the `Dialog (...) Button` at the end of the row to open the Variable Definition dialog. See Fault Handling Processing Rules.
4. Drag an activity into the catch handler, such as a reply or compensate. For an example, see Fault Handling for Service Invocations.
5. Fill in the properties for the activity to handle the fault.

To add the catch all handler for the process:

1. Click on the Fault Handlers tab of the Process Editor canvas.
2. Drag an **Error catch event** to the canvas.
3. Select the *Catch All Faults* checkbox.
4. Do not select any fault name or variable in the Properties view of the Catch.
5. Drag an activity into the catch all handler, such as an assign or compensate.
6. Fill in the properties for the activity to handle the fault.

The following illustration shown an example of a fault handler added for the process.
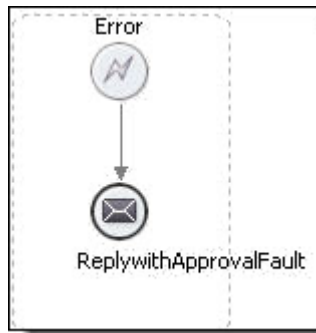
Tip: If you collapse the fault handler, you can add a background color from the Properties view. Right-mouse-click the handler and select Collapse Container.

Adding a Fault Handler for a Scope

You can add multiple catch handlers and one catch all handler for each scope in your process.

For explanations of these handlers, see Defining Catch and CatchAll Fault Handlers and Catching Undeclared and SOAP Faults.

To add a catch handler to a scope:

1. Select a scope container on the Process Activities tab of the Process Editor canvas.

2. Drag an **Error catch event** into the scope or near the border of a collapsed scope, as the illustration shows.
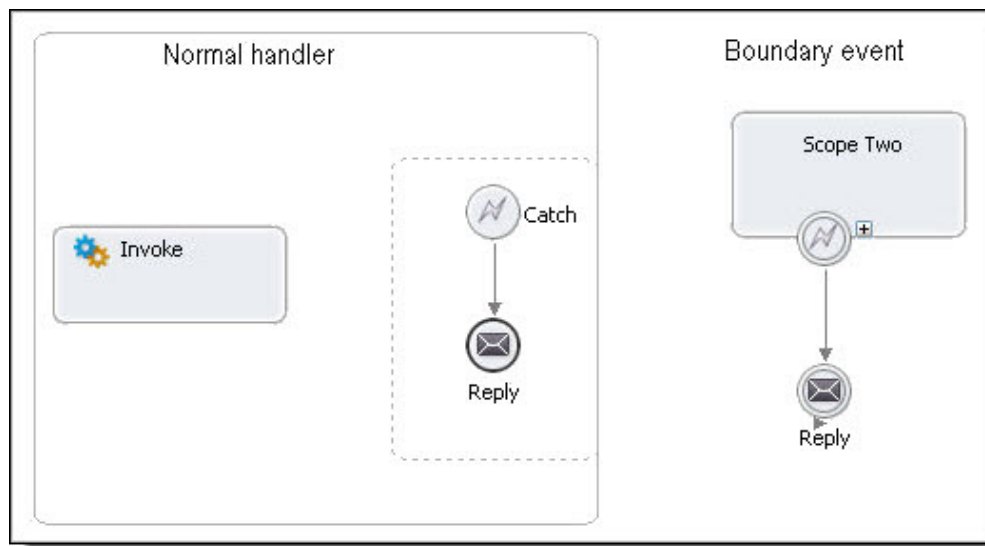
3. In the Properties view, select the following optional values:

   a. Select the **Catch All Faults** checkbox and leave the other properties blank.

   b. **Fault Name**. See Selecting a Fault Name for details. For a discussion on whether or not to add a fault name, see Fault Handling Processing Rules and Rules for Catching Faults in a Catch Activity.

   c. **Fault Variable**. Type in name for the variable described by the variable definition. For a discussion on whether or not to add a fault variable, see Fault Handling Processing Rules and Rules for

Catching Faults in a Catch Activity. If you have selected a Fault Name, the fault variables picklist displays only compatible variables.

    d.   **Variable Definition**. Select the `Dialog (...) Button` at the end of the row to open the Variable Definition dialog. See Adding a Fault Variable Definition for details. For a discussion on whether or not to add a a fault variable, see Fault Handling Processing Rules and Rules for Catching Faults in a Catch Activity.

4.   Drag an activity into the catch handler, such as a reply. For a boundary event, you must link the handler to an activity to execute. The activity must be an activity that has not yet executed.

5.   If you are using a boundary event, you cannot use certain activities. See Catch and Catch All Boundary Events and Compensate, Compensate Scope and Rethrow.

6.   Fill in the properties for the activity to handle the fault.

To add the catch all handler for a scope:

1.   Select a scope on the Process Activities tab of the Process Editor canvas.

2.   Drag an **Error catch event** into the scope or near its border.

3.   In the Properties view, select the *Catch All Faults* checkbox.

4.   Drag an activity into the catch all handler, such as an assign or compensate.
    If you are using a boundary event, you cannot use certain activities. See Catch and Catch All Boundary Events and Compensate, Compensate Scope and Rethrow.

5.   Fill in the properties for the activity to handle the fault.

The following illustration shows an example of Catch All fault handler added for a scope.
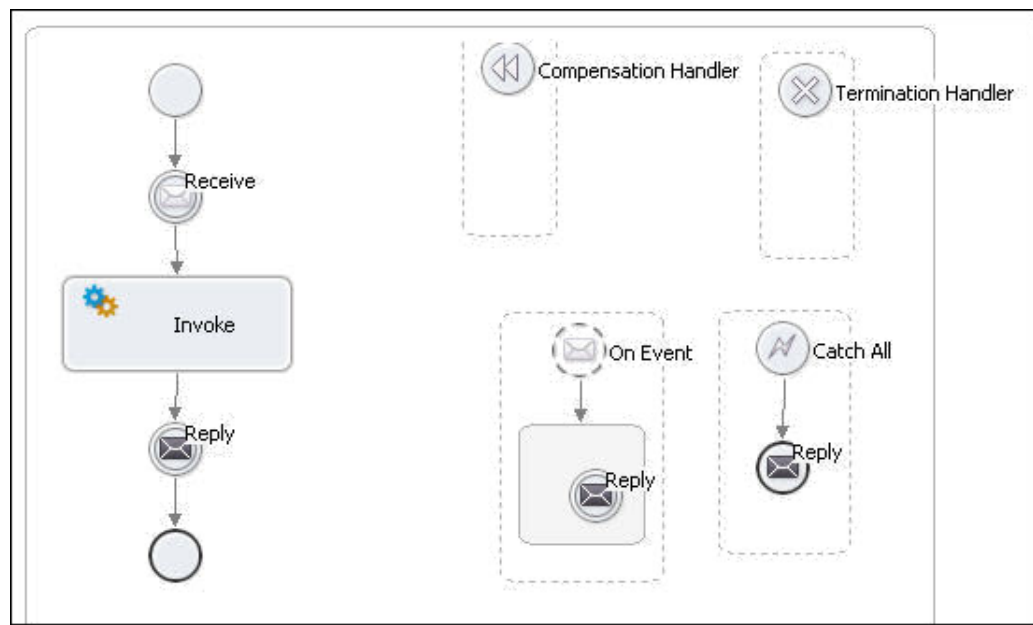
Tip: If you collapse the fault handler, you can add a background color from the Properties view. Right-mouse-click the handler and select **Collapse Container**.

Adding a Fault Handler as a Boundary Event for an Invoke Activity

When an invoke activity contains both an input and an output message, you can add a fault handler to catch any output message errors.

To add a fault handler to an invoke activity:

1. Display the Process Activities tab of the Process Editor canvas. You must be using the BPMN Process Editor in order to use boundary events.

2. Drag a Service task to the drawing canvas.

3. Add a fault handler (or event or compensation handler) to the border of the activity.

4. Follow the steps in Adding a Fault Handler for a Scope.

Selecting a Fault Name

Define a fault name based on a fault type and variable.

A catch or throw activity has a fault name property that you can open in Properties view. In the **Fault Name** dialog you can make the following selections:

- **Scope Faults**
  A scope fault name is typically from the WSDL for a service being invoked. The name of the fault is typically the name of fault on an operation with the target namespace as the qualifying namespace value. If your process uses a People activity (on-premises only), there are some standard Human Task faults included. Select a fault from the list.

- **Standard Faults**
  A standard fault is a one defined in the WS-BPEL 2.0 specification. Select a fault from the list. See also BPEL Standard Faults else where in this help.

- **Custom Faults**
  A custom fault is one you create and can include a prefix or namespace, if desired. You must use a custom fault name when catching undeclared faults.

The fault name is a required property of a throw activity and an optional property of a catch activity.

Adding a Fault Variable Definition

Select the variable definition, either message type or element. Then select a variable from the list.

Open the Fault Variable Definition dialog from the Properties view of a catch container.

1. Select **Message Type** for a list of all available messages.

2. Select Element if the message has a single part and that part is defined with an element type.

3. After you select a variable definition, return to the Properties view of the catch to type in a Variable name for the local catch variable.

After receiving an inbound fault message, the fault handler assigns the inbound fault message to the variable before proceeding to perform the activity enclosed by the catch. Since the variable is declared within a fault handler, it is not accessible outside of the fault handler much like a local variable is not accessible outside of its declaring scope.

Fault Handling Processing Rules

Fault Handling Processing Rules

When a fault occurs in a process, the flow of execution moves from the activity that generated the fault (for example, the invoke that faulted during execution or a throw activity) to the immediately enclosing scope's fault handler. If there is no immediately enclosing scope, the execution moves to the process's fault handler.

Once at the fault handler, a single `<catch>` or `<catchAll>` is matched in order to execute. The rules for matching a fault are as follows.

- There are several rules and a catch-matching priority scheme for executing a `<catch>`. For details, see *Rules for Catching Faults in a Catch Activity*.

- If fault cannot be matched to a `<catch>` using the rules referenced above, then it executes the `<catchAll>`.

- If there is no `<catchAll>` within the fault handlers, then the implicit fault handling logic executes. This logic acts like there is a `<catchAll>` present that contains a single compensate activity that executes the default compensation routine for all enclosed scopes. Once executed, the original fault is rethrown to the next enclosing scope or the process if none is available. If the handler is already at the process level, then the process terminates with the fault.

Prior to the execution of the matched `<catch>` or `<catchAll>`, all of the activities within the scope are terminated. Once a scope catches a fault, it is considered to have not completed normally and as such is not eligible for compensation for that execution. If the scope catches the fault without rethrowing the fault, then normal process execution can resume from the point of the scope on. If this happens at the process level, then the process completes normally but would not be eligible for process instance compensation.

In the following example, notice that the fault name and fault variable are not unique across catch activities.

Example

```
<faultHandlers>
 <!-- catch all faults with a matching name, but no data -->
  <catch faultName="x:foo">
    <empty/>
  </catch>
 <!-- catch all faults with the matching variable type,
       whose name is not "x:foo"-->
  <catch faultVariable="bar"
    faultMessageType="tns:barType>
    <empty/>
  </catch>
 <!-- catch the fault specified by the name
        and variable type -->
  <catch faultName="x:foo" faultVariable="bar"
     faultMessageType="tns:barType>
    <empty/>
  </catch>
 <!-- catch all faults not caught by a specific handler -->
  <catchAll>
     <empty/>
  </catchAll>
</faultHandlers>
```

The Rethrow Activity and Fault Handling

A BPEL process can rethrow an original fault caught by the nearest enclosing fault handler with a `<rethrow>` activity. A `<rethrow>` activity can be used within any fault handler. Regardless of how a fault is caught and whether a fault handler modifies the fault data, a `<rethrow>` activity always throws the original fault data and preserves its type.

Links and Fault Handling

A link that crosses a fault handler boundary must be outbound; that is, it must have its source activity within the fault handler and its target within a scope that encloses the scope associated with the fault handler.

Rules for Catching Faults in a Catch Activity

A catch activity can include a fault name and/or a fault variable. The fault variable can be defined as a WSDL message type or a schema element. The schema element can be a member of a substitution group defined in a WSDL's schema, or it can be a single-part message defined as an element.

The fault matching logic for WS-BPEL is a best match strategy. This means that the available catch elements are analyzed to determine which catch is the best match for the given fault. Contrast this with the fault matching logic for a language like Java which has a first match strategy. Because of the best match strategy, the order of the elements within the fault handlers does not affect which catch element is matched.

It is possible that more than one catch will match to a given fault. In this case, the catch that is matched with the higher priority rule is selected. In the case where two catch elements are selected by the same rule, then the best match is determined by whichever catch element declares an element variable that is closer to the fault element in terms of its substitution group hierarchy.

The priority system first considers whether a catch has a only a fault name or a fault name with data. The highest priority is for a catch with only a fault name and no data.

For catches with fault data, Process Developer executes catches, in the order described in the following table.

Catch Order	Fault handler option selected	Variable Type Definition	Data from the Fault Variable
Catching a fault with a fault name, but no variable data:			
1	Fault Name only	none	none
Catching a fault with variable data:			
2	Fault Name and Variable	message or element	message or element
3	Fault Name and Variable	element	element (substitution group)
4	Fault Name and Variable	element	single part element message
5	Fault Name and Variable	element	single part element message (substitution group)
6	Fault Name only	none	message or element is ignored but is rethrown to enclosing scope
7	Variable only	message or element	message or element
8	Variable only	element	element
9	Variable only	element	single part element message
10	Variable only	element	single part element message (substitution group)

For details on execution rules for catch/catch all, see Fault Handling Processing Rules.

Tips on Fault Handling

Note the following tips:

- The name of a declared WSDL fault and its associated data are unavailable to a `<catchAll>`. If you need access to this information, be sure to provide a specific `<catch>` with the fault name and fault variable.

- You can rethrow the fault that matched your `<catch>` or `<catchAll>` using the `<rethrow>` activity. The original fault data is rethrown, including the fault name and unmodified fault data.

- You are not limited to working with faults defined in a service's WSDL. You can define a `<throw>` activity to use a fault name and variable of your own creation.

- The standard runtime BPEL faults can occur due to errors in the design of a process (for example, having two receives executing concurrently on the same partner link and operation results in a `bpel:conflictingReceive`) as well as standard runtime errors. In general, it is not good design practice to catch these faults since they mostly represent errors that should be caught during simulation or testing of the process. There are some exceptions to this. For example, `bpel:joinFailure` is useful for identifying a situation where an activity does not execute due to its inbound links resulting to false.

Catching Undeclared and SOAP Faults

In standard fault handling, a fault message is declared as part of a WSDL operation. This declaration is a best practice and is expected behavior for BPEL fault handling. However, best practices do not always exist for Web services, and fault messages can be missing. When a service faults, the process faults with no information about the cause. To improve upon this behavior, Process Developer provides fault handling for undeclared faults.

To catch undeclared faults as well as SOAP faults, you can declare your own faults and then use the Process Developer fault custom functions to handle non-WSDL faults.

Using the Process Developer custom functions, you can prevent a process from terminating and can get access to the following fault information:

- `getFaultCode()`: Returns the category of the fault

- `getFaultString()`: Returns a short description related to the fault code

- `getFaultDetail()`: Returns information about what caused the fault. You can catch a specific Java fault, including the class name, message, and stack trace. As described below in Example 4, implement this feature by using a custom fault name in a catch. Note the required namespace for the qualified custom fault name for a Java fault.

- `getSOAPFault()`: Returns the entire `soap:Fault` element (if fault was due to a SOAP invoke that resulted in a fault).

Here are some examples of how to create fault handling of undeclared faults.

Example 1: Add a CatchAll handler for undeclared faults and SOAP faults

1. Add a catchAll to a scope or to the process, as described in Defining Catch and CatchAll Fault Handlers. (Or add the handler as a BPMN boundary event).

2. Add an assign activity as the target activity of the handler.

3. Copy an expression to a process variable using the fault custom functions, as shown in the following example:
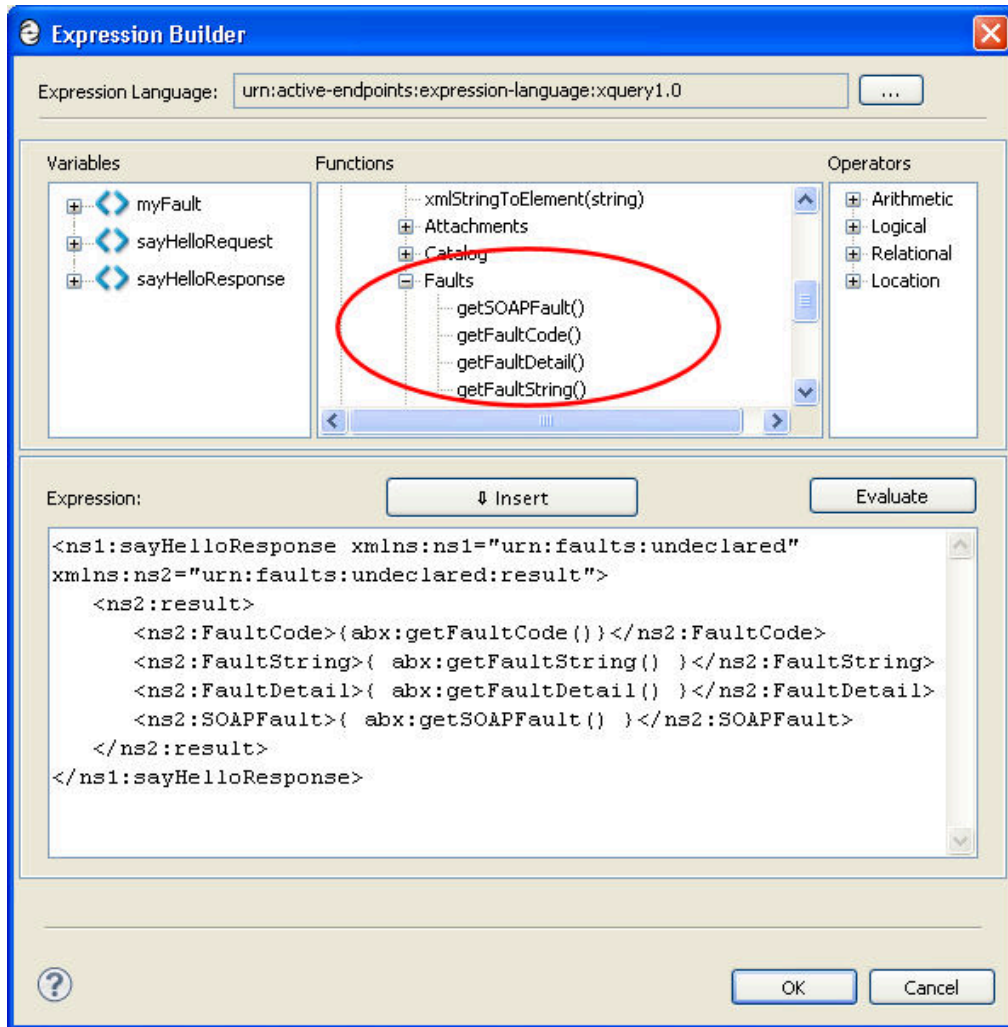
4. The copy operation you create would look like the following XQuery example:

   Copy From:

   ```
   <ns1:myFaultInfo xmlns:ns1="urn:faults:undeclared"
   xmlns:ns2="urn:faults:undeclared:result">
       <ns2:result>
          <ns2:FaultCode>{abx:getFaultCode()}</ns2:FaultCode>
          <ns2:FaultString>{ abx:getFaultString() }</ns2:FaultString>
          <ns2:FaultDetail>{ abx:getFaultDetail() }</ns2:FaultDetail>
          <ns2:SOAPFault>{ abx:getSOAPFault() }</ns2:SOAPFault>
       </ns2:result>
   </ns1:myFaultInfo>
   ```

   Copy To:

   ```
   myFaultInfo
   ```

   Example 2: Declare a process variable to add to a catch

1. Add a schema element to a project WSDL. This is used in a catch or throw fault variable definition. For example:

   ```
   <xs:element name="undeclaredFaultElement" type="tns:undeclaredFaultType"/>
       <xs:complexType name="undeclaredFaultType">
         <xs:sequence>
           <xs:element name="faultMessage" type="xs:string" />
         </xs:sequence>
       </xs:complexType>
   ```

2. Add a catch and use the schema element as a Fault Variable Definition:

```
Element = ns1:undeclaredFaultElement
```

3. Add a catch Fault Variable Name, such as *myFault*

4. Add an assign activity to the catch, and complete it as in Example 1.

Example 3: Declare a process variable to add to a throw activity

1. Add a schema element to a project WSDL, as in Example 2.

2. Add a new process variable, such as myFault, defined as the schema element.

3. Add an Initial Value to the variable as a literal, such as the following example:

```
<ns1:undeclaredFaultElement xmlns:ns1="urn:faults:undeclared">
   <faultMessage>string</faultMessage>
</ns1:undeclaredFaultElement>
```

4. Add a throw to the process.

5. Select myFault as the **Fault Variable**.

6. Create a Custom Fault Name for the throw, such as `ns1:myFault`.

Example 4: Catch a Java fault by name

1. Add the required namespac**e** to the process, as follows:

```
aex="http://www.active-endpoints.com/2004/06/bpel/extensions/"
```

This is the Process Developer namespace used for catching system errors.

2. Add a catch to the process.

3. Define a custom **Fault Name** as follows:

   - Prefix of `aex` or some other prefix that points to the required namespace shown in Step 1.

   - Unqualified class name of a Java exception. The name is the NCName of the class, without the package details. For example: `UnknownHostException`.
     Example: `aex:UnknownHostException`.

4. Complete the catch by adding an assign. See Example 1.

5. For example, if the `java.net.UnknownHostException` is thrown by an invoke due to an invalid URL, the fault detail element will appear as follows:

```
<ext:UnknownHostException xmlns:ext="http://www.active-endpoints.com/2004/06/bpel/
extensions/"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <Class xmlns="http://www.active-endpoints.com/2004/06/bpel/
extensions/">java.net.UnknownHostException</Class>
         <Message xmlns="http://www.active-endpoints.com/2004/06/bpel/
extensions/">unknown-host</Message>
         <StackTrace xmlns="http://www.active-endpoints.com/2004/06/bpel/
extensions/">java.net.UnknownHostException: unknown-host
 at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:177)
 at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:366)
 at java.net.Socket.connect(Socket.java:525)
 at java.net.Socket.connect(Socket.java:475)
 at java.net.Socket.<init>(Socket.java:372)
 ...
 at org.activebpel.work.AeDelegatingWork.run(AeDelegatingWork.java:62)
 at
org.activebpel.work.AeExceptionReportingWork.run(AeExceptionReportingWork.java:58)
 at org.activebpel.work.AeWorkerThread.run(AeWorkerThread.java:142)
   </StackTrace>
</ext:UnknownHostException>
```

CHAPTER 25

Simulating and Debugging

Simulating and debugging topics are as follows:

What is the Process Developer Debug Perspective

The Process Developer Debug perspective contains the views, editors, menus, and toolbars that support the execution of a BPEL process that you create or import into Process Developer.

During simulation, you can debug your process by using breakpoints or stepping. In simulate/run mode, the process executes but the execution cannot be suspended or examined. In simulate/step mode, execution can be suspended and resumed and variables can be inspected and modified.

For concepts and tips on using perspectives, see "Windows Perspectives Views and Editors" on page 43

For Debug perspective details, see:

Opening the Process Developer Debug Perspective

Process Developer automatically switches to the Debug perspective when you begin simulation or remote debugging. However, you can manually display the perspective in one of the following ways:

- From the Perspective bar on the top right of the Project Explorer, select the Process Developer Debug Perspective icon.

- From the Window menu, select **Open Perspective > Other > Process Developer Debug**.

Tip: You can open a perspective in the same window or in a new window. Open a perspective in a new window from the **Window > New Window** menu. There is also a Workbench preference for always opening a new perspective in a new window.

Switching Between Process Developer Perspectives

There are two Process Developer perspectives: Design and Debug. Each perspective has a set of views and menus supporting a set of related tasks.

Use one of the following ways to switch between perspectives:

- Click the icons on the Perspective bar.
- Select **Window > Open Perspective** and select a Perspective.
- If both perspectives are open, use the keyboard shortcuts for Next Perspective (Ctrl +F8) and Previous Perspective (Ctrl +Shift + F8).

Tips for displaying perspectives:

- The window titlebar displays the name of the currently active perspective, helping you manage open perspectives.
- If you have closed or rearranged views, you can reset the perspective to its default display by selecting **Window > Reset Perspective**.
- Select color and font preferences for the Debug perspective. See .

Process Developer Debug Perspective Views and Menus

The following illustration shows the views and menus that make up the Debug perspective:
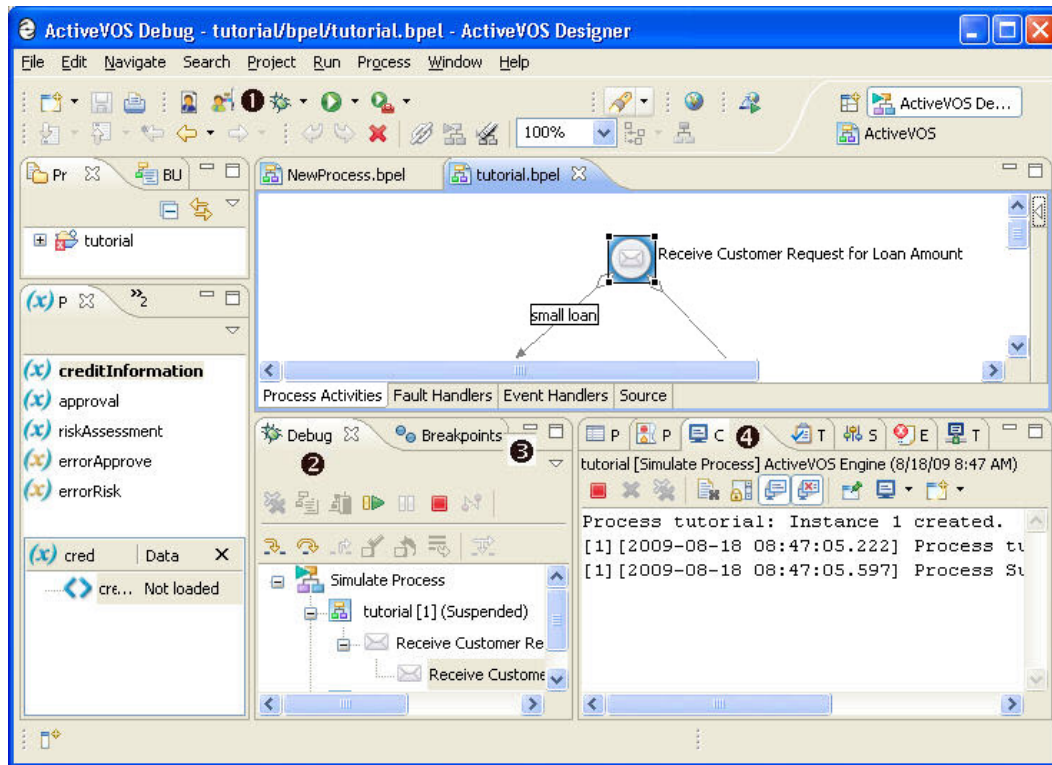
1	Simulate Process and Clear Execution State toolbar icons. See "Starting and Ending Simulation of a BPEL Process " on page 314 and "Clearing the Process Execution State " on page 317
2	Debug view. See "Using the Process Developer Debug View " on page 308
3	Breakpoints view. See "Using Breakpoints in BPEL Process Simulation " on page 310
4	Console. See "Using the Process Developer Debug Console " on page 312

Using the Process Developer Debug View

The Process Developer Debug perspective contains a Debug view that lets you manage the running or stepping through of a BPEL process that you have created or imported into Process Developer. It displays an execution tree associated with the process you are debugging.

The Process Developer Debug view is based on the Eclipse Debug view, and not all functions apply to Process Developer. You will notice that some icons, options, and preferences are unavailable in Process Developer.

The following illustration shows a sample Debug view. The table describes function keys available.

1	Resume (F8 key)	Resumes a suspended thread. Allow the process to run until the next breakpoint is encountered or until the process is completed.
2	Terminate	Terminates the selected debug target
3	Step Into (F5 Key)	The process executes until the next activity in the process is reached. If the activity is a container, such as a scope, if, while or for each, the execution of each activity in the container takes place. **Tip:** You can also step into an invoke activity that is a subprocess. For details, see "Selecting an Invoke Subprocess for Simulation " on page 322.
4	Step Over (F6 Key)	The process executes until the next activity in the process is reached. If a breakpoint is encountered, the execution suspends at the breakpoint.
5	Step Return (F7 Key)	Use this to return from a container that has been stepped into. Even though the step returns from the container, the remainder of the activities in the container are still executed.

In addition to the toolbar options, there are more options on the right mouse menu of a simulation thread item. The following illustration shows only the additional options, not all the options available on the right mouse menu.
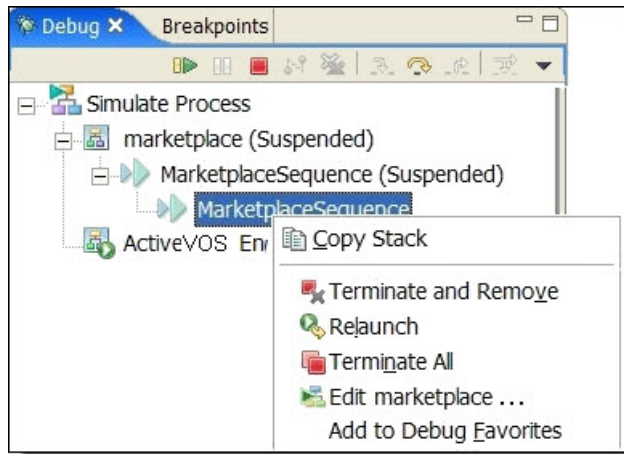
Copy Stack	Copies the selected stack of suspended threads as well as the state of the running threads to the clipboard
Terminate and Remove	Terminates the selected debug target and removes it from the view
Relaunch	Start a new simulation
Terminate All	Terminates all active simulations in the view

Using Breakpoints in BPEL Process Simulation

The Breakpoints view lists all the breakpoints you have set in BPEL processes. You can double-click a breakpoint to display its location in the Process Editor. In the Breakpoints view, you can also enable, disable, skip, or remove breakpoints.

The following illustration shows a sample Breakpoints view and process. The breakpoints were added next to an activity, and the list of breakpoints appears in the Breakpoints view.
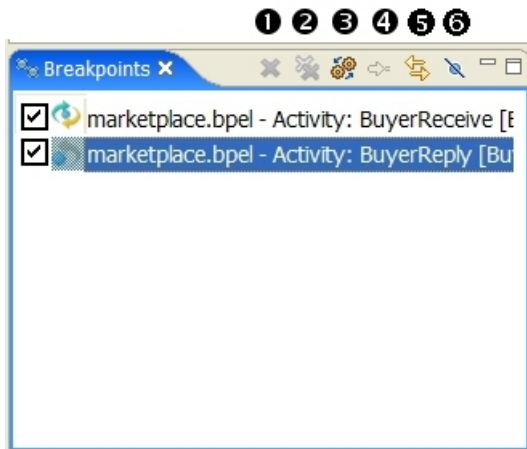
1	Removes selected breakpoints
2	Removes all breakpoints for all targets
3	Shows breakpoints supported by selected target
4	Goes to file for breakpoint
5	Ties the view to always show the current file being debugged
6	Skips all breakpoints

Adding a breakpoint:

1. From the Project Explorer, open a BPEL file.

2. Right-mouse click on an activity where you want to set a breakpoint and select **Add Breakpoint**.

A red circle appears next to the activity to indicate the breakpoint, as shown.
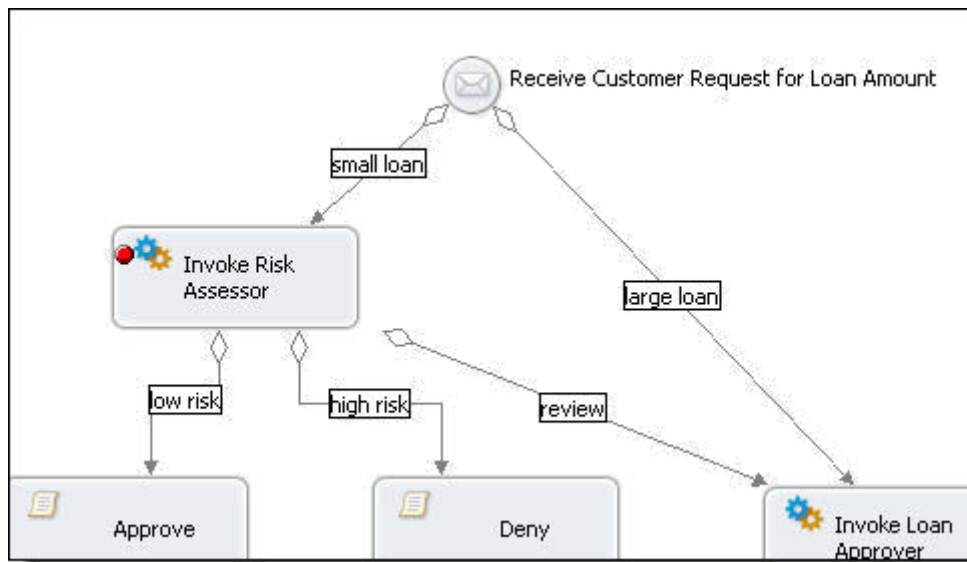
Tips for working in the Breakpoints view

- Select **Remove All Breakpoints** from the toolbar to delete all breakpoints displayed.

- Double-click a breakpoint to highlight its location in the Process Editor.

- Right-click a breakpoint and select **Go to File** to go to the activity in the Process Editor where the breakpoint is set. If the file is not already open, this option opens the file and finds the breakpoint.

- Each breakpoint shows the name of the BPEL file and activity where the breakpoint is set. All breakpoints are listed for all files you are debugging.

- You can **Enable** and **Disable** breakpoints by selecting/deselecting the check mark in the box next to the breakpoint. A disabled breakpoint appears as a grey circle next to an activity icon.

See also "Running to a Breakpoint in a BPEL Process " on page 315.

Using the Process Developer Debug Console

The Console view is part of the Process Developer Debug perspective.

This view shows execution events and provides a view into the evaluation of expressions and results.

The following illustration shows a sample of the Console.
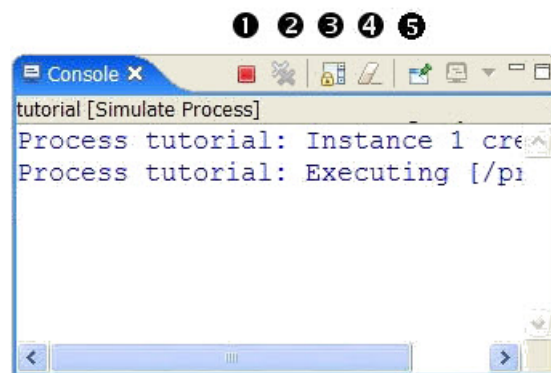
1	Terminate. Stops the current simulation
2	Removes all terminated launches
3	Scroll Lock. Automatically wraps long lines to avoid horizontal scrolling.
4	Clears console
5	Pin Console. Not applicable.

As your process executes, the following types of events are displayed:

Activity type, name, and path	Executing, completed normally, or completed with a fault Example: `Receive:Executing </process/flow/receive>`

Link name, link transition condition, and path	Status and evaluation of expression
	Example:
	`Link receive-to-assess Condition true :`
	`bpws:getVariableData('request','amount') < 10000 </process/flow/`
	`links/link[@name='receive-to-assess']>`

On Alarm	Duration or deadline
Wait	Wait value and simulated Wait value
Join Condition	Evaluation of expression
While, If, Repeat Until Condition	Evaluation of expression
Unhandled events	Ancillary event information

Tips for using the Console:

- Use the right-mouse menu to go to an activity that has already executed.

- Use the right-mouse menu to find a simulation event in the Console.

Simulating Execution of a BPEL Process

After designing but before deploying a BPEL process to a server, you can simulate execution in the Process Developer.

To simulate, you must have a valid process, and you must supply certain input and output sample data values to simulate data that a running process would normally receive and send.

Ensure you are ready to simulate execution of your process. See "Prerequisites for Simulation " on page 314. You can start simulation from either the Process Developer or the Debug perspective.

Select a topic for more information:

- "Starting and Ending Simulation of a BPEL Process " on page 314

- "Running to a Breakpoint in a BPEL Process " on page 315

- "Stepping to the Next Activity in a BPEL Simulation " on page 316

- "Viewing the Execution State of an Activity or Link " on page 316

- "Modifying a BPEL Process During Simulation " on page 317

- "Terminating and Removing BPEL Process Simulations " on page 317

- "Clearing the Process Execution State " on page 317

Prerequisites for Simulation

Before starting simulation, do the following:

- Ensure that you have filled in all the required properties for each activity, such as partner link, operation and variable (or `fromPart`) for receives, picks, invokes (or `toPart`), and replies

- Ensure that the Abstract Process property is set to *No* for the process. The default is *No*.

- Make sample data available for the input, fault, and output variables that a running process would receive and send. For more information, see "Supplying and Inspecting Sample Variable Data During Simulation " on page 318.

- If your process contains a Pick activity, you can select a branch to execute. For more information, see "Selecting Simulation Paths and Properties " on page 321.

Starting and Ending Simulation of a BPEL Process

To simulate your process:

1. Open your BPEL file.

2. Click your mouse in the Process Editor canvas to activate the editor toolbar.

3. Select the **Simulate Process** icon on the toolbar, as shown.
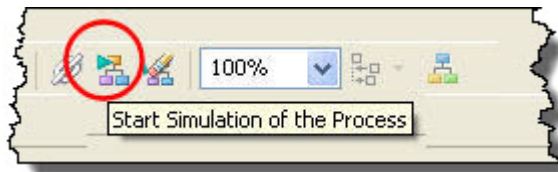
The simulation target stops at the first activity in the process and suspends execution. If you have not added sample data you are prompted to add it.

4. In the Debug view, do one of the following:

   - Click the **Resume** icon to run to completion or to a breakpoint

   - Click the **Step Over** icon (F6 key) to step to the next activity or breakpoint

   - Click the **Step Into** icon (F5 key) to step into a container such as a while, if, or for each

As the process executes, the following events occur:

- Each activity is highlighted as it prepares to execute

- If necessary, messages appear to instruct you to add missing sample data

- In Process Variables view, sample data is cleared at the beginning of simulation and then is displayed as the corresponding activities execute

- The Console view shows simulation events

- The Outline view shows execution of all activities, some of which can be hidden on the canvas

- If an error occurs, an error message pops up, the simulation terminates, and the activity is marked with an X, as shown in the following illustration. For more information, see "Inspecting Standard Faults During Simulation " on page 324.
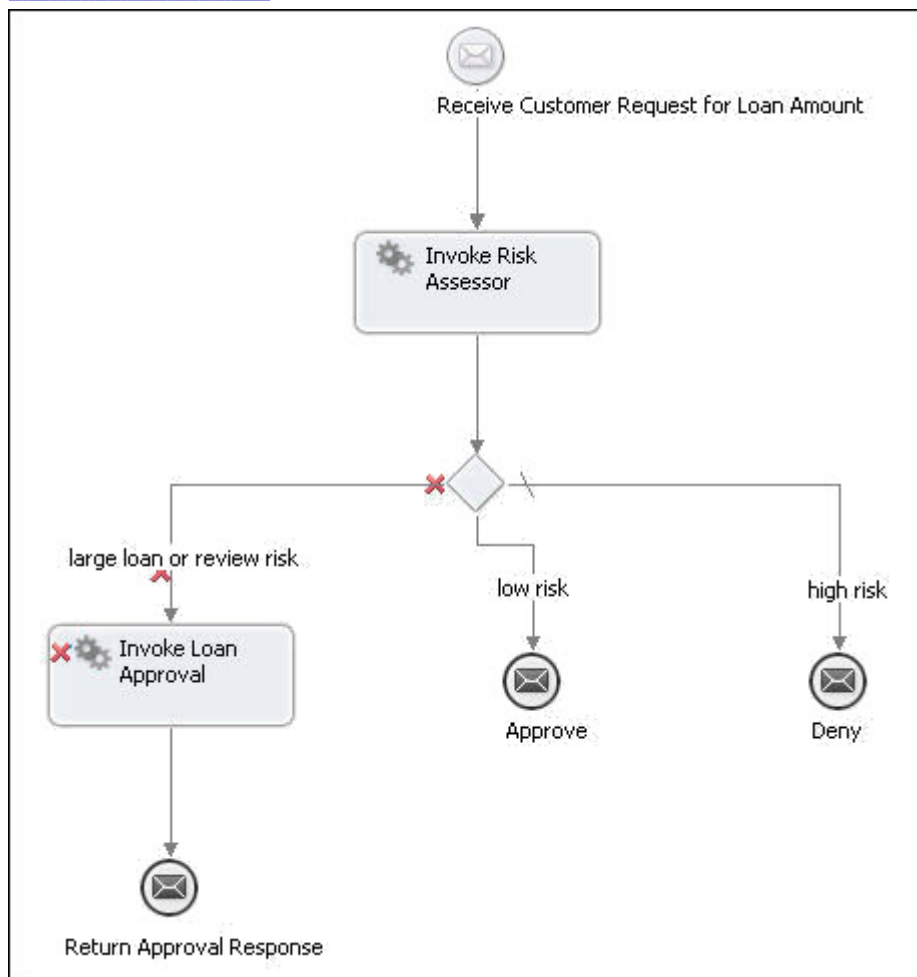
You can end a simulation thread by stepping to the end or by terminating it.

You can select **Terminate** from the context menu of the process's thread in the Debug view to terminate the simulation.

When a simulation terminates, the process diagram shows the simulation path. The illustration above shows both the path taken (highlighted) and not taken (grayed out).

Running to a Breakpoint in a BPEL Process

When you are debugging a BPEL process, the simulation suspends before the first activity is executed.

You can resume execution and run to a breakpoint that you have set next to an activity in a process. See "Using Breakpoints in BPEL Process Simulation " on page 310 for information on setting breakpoints.

To run to a breakpoint:

1. Select a process in the Debug view.

2. Click the **Resume** button in the Debug view toolbar (or press the F8 key). The process resumes its execution.

Stepping to the Next Activity in a BPEL Simulation

You can step through the execution of a BPEL process one activity at a time. While you are stepping through the simulation, you can see execution results in the Console. You can also follow the execution path on the canvas as well as the Outline.

Click the **Step Over** button in the Debug view toolbar, or press the F6 key. The currently-selected activity is executed and suspends on the next executable activity. Alternately, click **Step Into** (F5 key), which steps into a container, such as a while, if, or for each.

The following illustration shows one step at a time of an execution path.
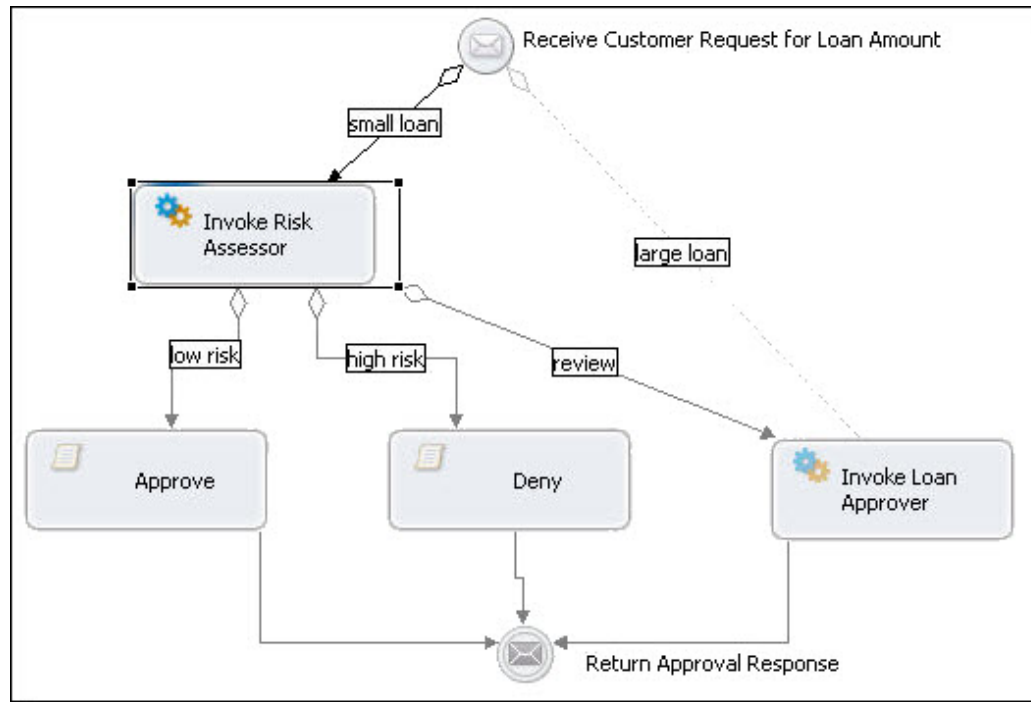
Tips:

- If a breakpoint is encountered during a step operation, the execution suspends at the breakpoint, and the step operation ends.
- If you have used the input, output, or data tab of the Receive, Invoke, Reply activities to map data, use the Outline view to view the progress of step into. These activities have generated scopes with assign activities that are not shown on the canvas, only on the Outline.

Viewing the Execution State of an Activity or Link

- Inactive
- Active
- Ready to execute
- Executing
- Dead Path (will not run because of links or conditional execution)
- Execution Completed (activity only)
- Execution Faulted (activity only)

You can view the execution state in the Properties view for the activity or link.

Modifying a BPEL Process During Simulation

It is possible for you to make changes to the BPEL process during simulation. For example, you can change sample data values, expressions, correlation property values, and partner link addresses.

For details, see:

- "Supplying and Inspecting Sample Variable Data During Simulation " on page 318
- "Updating Correlation Property Data " on page 340
- "Updating Partner Link Address Information " on page 341
- "Correcting Retrying or Completing Activities " on page 336

The simulator determines whether or not the change causes the process model to be out-of-sync with the process currently being executed.

Terminating and Removing BPEL Process Simulations

Use one of the following options to stop a simulation and to remove a simulation from the Debug view.

Option	Location	Description
Terminate	Debug view toolbar and Run menu Right mouse menu of simulation thread Console toolbar	Terminates the simulation of the associated process
Terminate All	Right mouse menu of simulation thread	Terminates all active simulations
Terminate and Relaunch	Right mouse menu of simulation thread	Terminates the simulation of the associated process and relaunches it
Terminate and Remove	Right mouse menu of simulation thread	Terminates the simulation of the associated process and removes it from the view
Remove All Terminated	Debug view toolbar Right mouse menu of simulation thread	Clears all terminated simulations from the view

Clearing the Process Execution State

When you terminate a BPEL process simulation, the highlighting that had been added to activities and links remains in view in the Process Editor and the simulation state of each activity and link is displayed in Properties view.

To remove highlighting and simulation states, click the **Clear Execution State** icon, as shown.
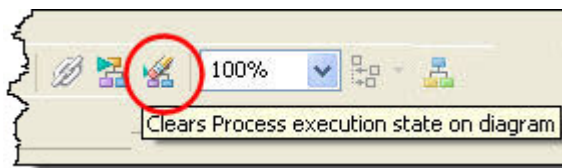
Supplying and Inspecting Sample Variable Data During Simulation

Before beginning simulation, you must initialize certain input and output variables with sample data values. The idea is to simulate the actual data that a running process would normally receive and send. Namely, you add sample data values for the following variables:

- Variable in the receive or onMessage activity that starts the process
- Output variable in the invoke activities. This is the data sent back to the process from an invoked Web service.
- OnEvent variable in an event handler
- Fault variables that you want to simulate

Process Developer offers several ways for you to supply sample data values for process variables. When you simulate process execution, you can use different data values to test different execution paths in your process.

Add, load, and change sample data as follows:

Add sample data files for message data	In Participants view, you can view WSDL messages and then add one or more sample data files per message. One data file per message defines default values for the message parts. For a discussion of adding sample data, see "Using Sample Data for WSDL Messages" on page 93.
Select default values for message data	If you add multiple data files to messages, you can specify one file as the default. See "Selecting a Default Sample Data File" on page 95.
Change default values for process variables used in receive, invoke, onMessage, and onEvent activities	As you simulate execution of a process, you can override the default data values for input, output, and fault messages. For more information, see "Setting up Sample Data Values for Input Output and Fault Messages " on page 318.
Load sample data values to initialize process variables	For a specific BPEL process, you can type in data values or load a data file for each process variable. For more information, see "Using Sample Data in Process Variables View" on page 212.

During process simulation, you can view variable assignments. For more information, see "Inspecting Process Variables during Simulation " on page 320.

Setting up Sample Data Values for Input Output and Fault Messages

Select the source location for the sample you want to add.

You can simulate execution with different data values that might be received by the process from receive, invoke, onMessage, and onEvent activities. You can load sample values from either the Participants or Process Variables view or the Properties view of an activity.

If you have created sample data files in the Project Explorer, in the sample-data folder, you can select a data file during simulation. For a message that is missing sample data, a Sample Data dialog displays appropriate selection, as shown in the example.
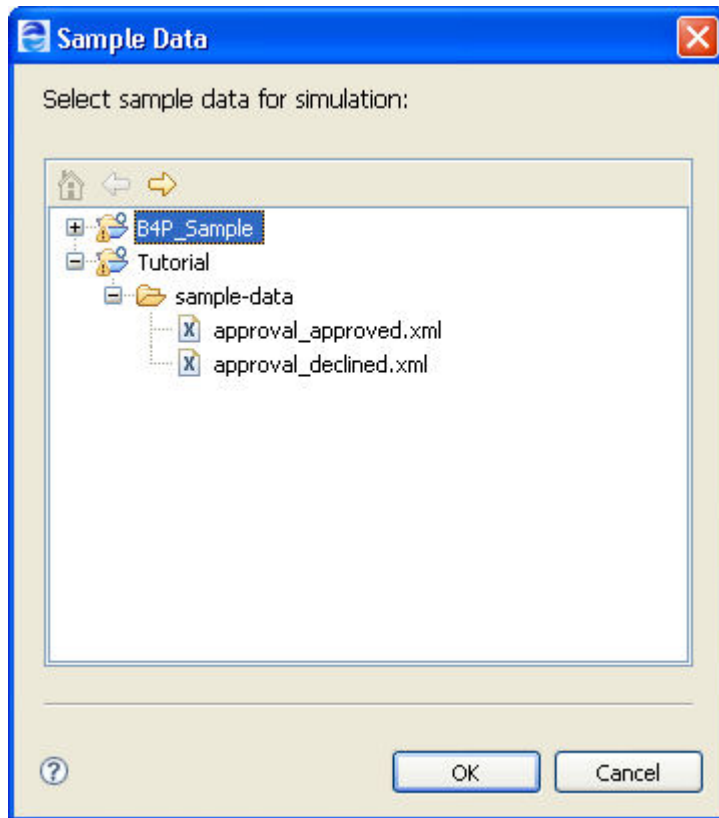
You can load sample data before you begin simulation. For information on loading and saving sample data in the Participants or Process Variables view, see "Using Sample Data in Process Variables View" on page 212.

During simulation, you can override data values on-the-fly for any activity that has not yet executed.

To override loaded sample data for an input and output messages:

1. From the Process Editor canvas, select a Receive, Invoke, onMessage, or onEvent activity.

2. In the Properties view, select the All tab and do one of the following:

   - For a receive, onMessage, or onEvent, click **Dialog (…)** for the Input Message or From Part row

   - For an invoke, click **Dialog (…)** for the Output Message or To Part row

3. In the **Set Simulation Data** dialog, do the following:

   - Select the checkbox next to Override sample data for simulation

   - Select the Dialog Button at the end of a row, as shown.
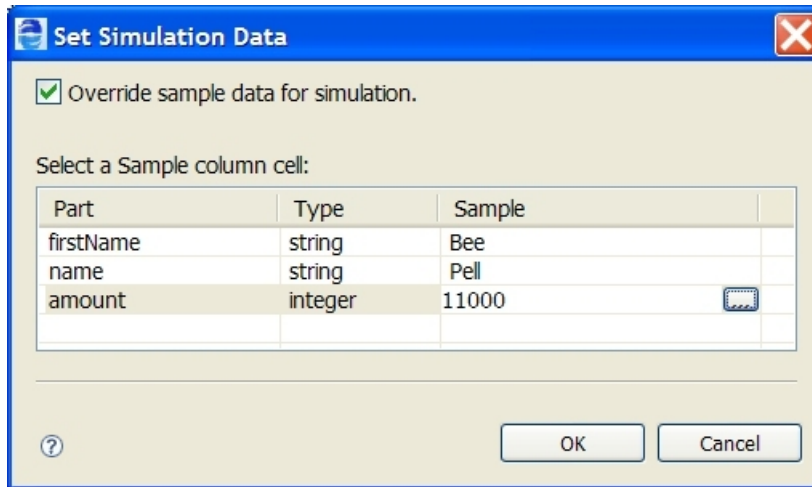
4. For simple data types, edit the data value in the Sample dialog and click **OK**. For complex message types, load a data file.

5. Edit any additional values for other message parts and click **OK.**

To override loaded sample data for an invoke fault message:

1. From the Process Editor canvas, select an Invoke activity that has an output and a fault message defined.

2. In the Properties view, set the Result property to Fault.

3. Select a Fault Name from the list.

4. Select a Fault Message, and click **Dialog (...)** .

5. In the **Set Sample Data** dialog, type in a value for the fault message.

Inspecting Process Variables during Simulation

During simulation, as each activity executes, you can inspect variable values in the Process Variables view. The values are those you loaded into Process Variables or into the simulation properties for receives, invokes, onMessage branches of picks and onEvent branches of event handlers. For information on loading and changing data values, see "Supplying and Inspecting Sample Variable Data During Simulation " on page 318.

Here is a recommended way to inspect variable values, by opening and closing variables as you need to.

1. Display the Process Variables view.
   Tip: Display the list of variables with all variables closed. To close all variables, select a variable from the list, and select **Close All** from the right-mouse menu.

2. Start simulation. For details, see "Simulating Execution of a BPEL Process " on page 313.

3. Step to the first receive or onMessage branch of a pick activity. For more information, see "Stepping to the Next Activity in a BPEL Simulation " on page 316.

4. In Process Variables view, double-click the highlighted variable to open it.

5. Right-mouse click the variable and select **View Data**.

6. Notice that the data view of open variable that is in use shows the current data.
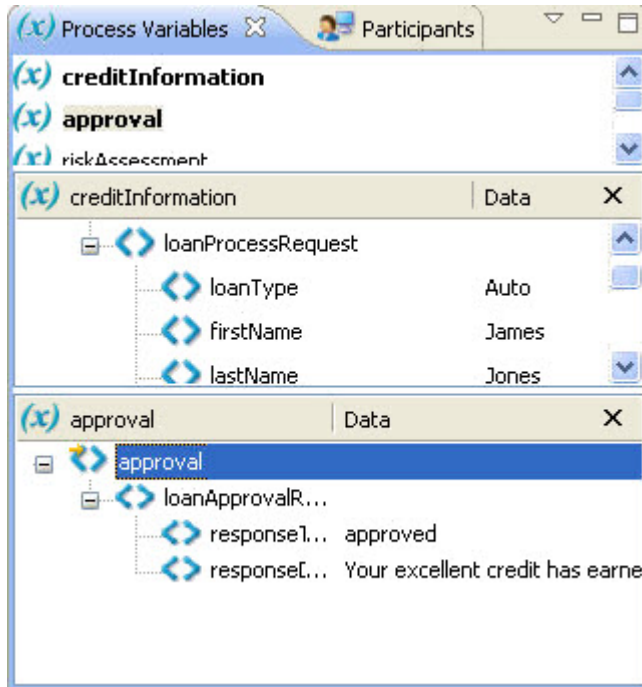
7. Step to the next activity to execute the current activity. As you step through, open variables in data view to inspect the current value.

Selecting Simulation Paths and Properties

Add an endpoint reference address to simulate with.

You can modify certain BPEL elements to cause different simulation events to occur.

For example, you can change a data value in a receive message variable to cause a link transition to be true or false. You can also change sample data values for output and fault messages.

Similarly, you can update the value of correlation properties, if needed, and also update the address information for a partner link.

The following table lists the simulation property for each BPEL element that you can modify.

Activity	Simulation Property	Description
Receive	Input Message Input attachments	Set the instance data for the simulated message to receive
Pick	Execute	Select the onMessage or onAlarm branch to execute for simulation. You must select an option before you simulate a Pick.
onMessage branch of pick or onEvent Event Handler	Input Message	Set the instance data for the simulated message to receive

Activity	Simulation Property	Description
Invoke	Output Message Output attachments	Set the instance data for the simulated message to return
	Result	Set the result of the output message to Normal or Fault
	Fault Name	When the Result is set to Fault, you can select a Fault Name from the list. Fault names are defined in the associated WSDL file.
	Fault Message Fault Attachments	Set the instance data for the simulated Fault message.
	Subprocess	For details, see "Selecting an Invoke Subprocess for Simulation " on page 322.
Wait	Wait in Seconds	Actual wait time in seconds for simulation. The activity wait time can be longer than you want to wait during simulation. This is a way to shorten that time.
Partner Link	My Role and Partner Role	Endpoint reference address that is used in a dynamic assignment within a Copy operation

The following table lists the execution property for each BPEL element that you can modify. Note that the Execution properties are only available during simulation or remote debugging and do not normally appear in the Properties view.

Element	Execution Property	Description
Correlation Set	Correlation Set Data	Update the current value of the correlation property
Partner link	Address	Add or update address information for the partner link

For details on scenarios using correlation sets, see Chapter 30, "Process Exception Management" on page 420.

Selecting an Invoke Subprocess for Simulation

Select a different process to synchronize with the process deployed to the server. If this invoke activity is defined with a BPEL process, you can step into it during simulation.

A subprocess is a BPEL process that is invoked by another BPEL process. You can create one BPEL process, deploy it, and then use the resulting service's operation to create an invoke activity.

During simulation, you can use the invoke activity's simulation property, subprocess, to select a BPEL process. By making this selection, you can step into the subprocess and simulate it when you step to the invoke activity. For details on creating a subprocess, see *Creating a BPEL Process as a Service for Another BPEL Process*.

To select an invoke subprocess for simulation:

1. Select an invoke activity that calls a BPEL process.

2. In the Properties view, select All.

3. In the Simulation category, select Subprocess.

4. At the end of the row, click the Dialog button to open the **Process Selection** dialog.

5. Select a process from the workspace.

If the sub-process has a process-level compensation handler, be sure to set a breakpoint on an activity in the compensation handler. If you do not have a breakpoint set on the compensation handler, simulation steps over it rather than into it.

Note that during simulation, depending on your process logic, you can go back and forth for execution steps between the main process and the subprocess. In the Debug view, select the execution thread that you wish to execute next.

Simulating Event Handlers

During simulation, there are different debug targets for the main process and for event handlers. You can step into an event handler at any point in the simulation of its scope, and then select the main target to continue simulating the scope in the main process.

The following illustration shows the simulation targets:
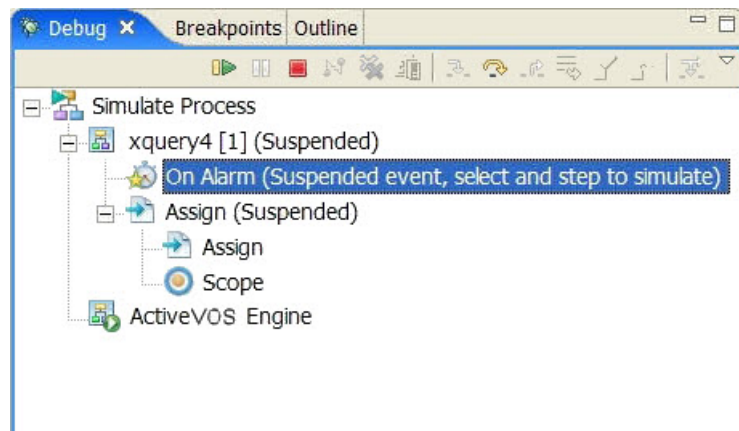
Simulating Fault Handlers

During simulation, you can execute a fault handler by setting a simulation property for the relevant invoke activity.

In the Properties view, set the **Result** property to **Fault** and then select the appropriate fault name and variable.

For more information, see "Setting up Sample Data Values for Input Output and Fault Messages " on page 318.

Inspecting Standard Faults During Simulation

The WS-BPEL specification defines several standard faults. If one of these faults is encountered during simulation, the simulation terminates and shows a red X next to the activity where the fault occurred. Also, you see a listing for the fault in the Console. An example is:

```
Assign AssignYestoAccept:
        Completed with fault: mismatchedAssignmentFailure :
                </process/flow/assign[@name='AssignYestoAccept']>
```

Simulation Preferences

Set preferences for variable handling during simulation.

When you simulate execution of a process in Process Developer, the simulation engine behaves identically to the server engine, where your deployed processes run. You can set preferences to modify the server engine behavior, and you can set the same preferences for the simulation engine.

To learn more about setting server engine behavior, refer to the Process Server documentation.

Select **Window > Preferences > Process Developer > Simulation** to view simulation preferences, described in the following table.

Simulation Preference	Description
Validate input/output messages against schema	Validates the sample data loaded into process variables against the WSDL schema. If you add a sample data file to a complex message, and the data file is not valid, Process Developer warns you and allows you to add it with errors.
	Enable this option to validate data before simulation starts. Disable this option for faster simulation. This option is enabled by default.
Disable bpws:selectionFailurefault BPEL Version 1.1 only	Enabling this option allows a null value to be returned from a function or assignment that contains an XPath (or other language) query string. You can enable this to override behavior, for cases that handle data samples with optional elements.
	By default, this option is not enabled, and if the query string returns an empty selection from an assign copy FROM, the process throws a bpws:selectionFailure fault, which is the standard response described in the BPEL4WS specification.
	To set a preference to include this option as a process extension for all WS-BPEL 2.0 processes, see *Process Developer Preferences*.
	See Disable *bpel:selectionFailure Fault Example* and *Disable bpel:selectionFailure Fault and Auto Create Target Path for Copy/To Example* for more information.
Auto create target path for Copy/To BPEL Version 1.1 only	Determines if Process Developer is allowed to create a location path for a non-existent node in a complex variable in a process instance document. When an assignment refers to a non-existent node (or to more than one node), the standard BPEL fault, bpws:selectionFailure, must be thrown, according to the BPEL specification.
	Enabling this option allows selections to be created on-the-fly. This means an assign copy TO operation can refer to a non-existent node and assign a value to it. This option is disabled by default.
	To set a preference to include this option as a process extension for WS-BPEL 2.0 processes, see *Process Developer Preferences*.
	See *Auto Create Target Path for Copy/To Example* and Disable bpel:selectionFailure Fault and Auto Create Target Path for Copy/To Example for more information see *Disable bpel:selectionFailure Fault and Auto Create Target Path for Copy/To Example*

Disable bpel selectionFailure Fault Example

The following example shows the effect of enabling and disabling the `Disable bpel:selectionFailure Fault` (BPEL4WS 1.1 processes: `bpws:selectionFailure`) option.

For details on WS-BPEL 2.0 processes, see *Using the Process Developer Disable Selection Failure Fault Extension*.

For a description of this BPEL4WS 1.1 preference, see *Simulation Preferences*.

Code sample:

```
<assign>
  <copy>
    <from part="OrderInfo" query="/ns1:OrderInfo
        /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
        variable="var1"/>
    <to part="OrderInfo" query="/ns1::Addr1"
        variable="var2"/>
  </copy>
</assign>
```

In the code sample, the `Assign From/To` query is for an optional element, as shown in the schema snippet below.

Schema Snippet for Var1 and Var2:

```
<xs:complexType
    name="AddressInfoType">
  <xs:sequence>
   <xs:element ref="ord:Name"/>
   <xs:element ref="ord:Addr1" minOccurs="0"/>
   <xs:element ref="ord:Addr2"
       />
   <xs:element ref="ord:City"/>
   <xs:element ref="ord:St"/>
   <xs:element ref="ord:Zip"/>
   <xs:element ref="ord:Cntry"
       />
  </xs:sequence>
 </xs:complexType>
 <xs:element name="BillToInfo"
    type="ord:AddressInfoType"/>
```

Var1 Sample Data	Var2 Initialization
```<ns1:OrderInfo xmlns:ns1="http://temp.com"> <ns1:OrderHeader> <ns1:OrdId>78</ns1:OrdId> <ns1:BillToInfo> <ns1:Name>Name1</ns1:Name> (Addr1 is missing) <ns1:Addr2>1 Main St </ns1:Addr2> <ns1:City>Albany</ns1:City> <ns1:St>NY</ns1:St> <ns1:Zip>12012</ns1:Zip> <ns1:Cntry>USA</ns1:Cntry> </ns1:BillToInfo> ...```	```<ns1:OrderInfo xmlns:ns1="http://temp.com"> <ns1:OrderHeader> <ns1:OrdId/> <ns1:BillToInfo> <ns1:Name/> <ns1:Addr1/> <ns1:Addr2/ <ns1:City/> <ns1:St/> <ns1:Zip/> <ns1:Cntry/> </ns1:BillToInfo> ...```

With `Disable bpel:selectionFailure Fault` *disabled*, the simulation ends in a Selection Failure fault because `Var1` is missing the query selection node, and it cannot be assigned to the `Var2` query selection node.

With `Disable bpel:selectionFailure Fault` enabled, the process terminates normally because the empty selection node is allowed. A null value is added to `Addr1` during the assignment.

Auto Create Target Path for Copy To Example

The following example shows the effect of enabling and disabling the Auto Create Target Path for Copy/To option.

For details on WS-BPEL 2.0 processes, see "Using the Process Developer Create XPath Extension" on page 104.

For a description of this BPEL4WS 1.1 preference, see "Simulation Preferences " on page 324.

Code sample:

```
<assign>
  <copy>
    <from part="OrderInfo" query="/ns1:OrderInfo
        /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
        variable="var1"/>
    <to part="OrderInfo" query="/ns1:OrderInfo
```

```
              /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
            variable="var2"/>
        </copy>
    </assign>
```

In the code sample, the Assign From/To query is for an optional element, as shown in the schema snippet below.

Schema Snippet for Var1 and Var2:

```
<xs:complexType name="AddressInfoType">
  <xs:sequence>
   <xs:element ref="ord:Name"/>
   <xs:element ref="ord:Addr1" minOccurs="0"/>
   <xs:element ref="ord:Addr2"/>
   <xs:element ref="ord:City"/>
   <xs:element ref="ord:St"/>
   <xs:element ref="ord:Zip"/>
   <xs:element ref="ord:Cntry"/>
  </xs:sequence>
 </xs:complexType>
 <xs:element name="BillToInfo" type="ord:AddressInfoType"/>
```

Var1 Sample Data	Var2 Initialization

<pre><ns1:OrderInfo
    xmlns:ns1="http://
    temp.com">
 <ns1:OrderHeader>
  <ns1:OrdId>78</ns1:OrdId>
   <ns1:BillToInfo>
    <ns1:Name>Name1</ns1:Name>
    <ns1:Addr1>Apt 12</ns1:Addr1>
    <ns1:Addr2>1 Main St
        </ns1:Addr2>
    <ns1:City>Albany</ns1:City>
    <ns1:St>NY</ns1:St>
    <ns1:Zip>12012</ns1:Zip>
    <ns1:Cntry>USA</ns1:Cntry>
   </ns1:BillToInfo>
...</pre>
	<pre><ns1:OrderInfo
	    xmlns:ns1="http:/
	    /temp.com">
	 <ns1:OrderHeader>
	  <ns1:OrdId/>
	   <ns1:BillToInfo>
	    <ns1:Name/>
	    (Addr1 is missing)
	    <ns1:Addr2/>
	    <ns1:City/>
	    <ns1:St/>
	    <ns1:Zip/>
	    <ns1:Cntry/>
	   </ns1:BillToInfo>
	...</pre>

With Auto Create Target Path for Copy/To disabled, the simulation ends in a fault because `Var1` contains the query selection node, but it cannot be assigned to the`Var2` query selection node because the location path is missing.

With Auto Create Target Path for Copy/To enabled, the process terminates normally because the location path for `Addr1` is built, and the value is added to `Addr1` during the assignment.

Disable bpel selectionFailure Fault and Auto Create Target Path for Copy To Example

The following example shows the effect of enabling both Disable bpel:selectionFailure Fault (BPEL4WS 1.1 processes: `bpws:selectionFailure)` and Auto Create Target Path for Copy/To options.

For details on WS-BPEL 2.0 processes, see *Using the Process Developer Create XPath Extension* and *Using the Process Developer Disable Selection Failure Fault Extension*.

For a description of these BPEL4WS 1.1 preferences, see *Simulation Preferences*.

Code sample:

```
<assign>
  <copy>
```

```
        <from part="OrderInfo" query="/ns1:OrderInfo
            /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
             variable="var1"/>
        <to part="OrderInfo" query="/ns1:OrderInfo
            /ns1:OrderHeader/ns1:BillToInfo/ns1:Addr1"
             variable="var2"/>
    </copy>
 </assign>
```

In the code sample, the Assign From/To query is for an optional element, as shown in the schema snippet below.

Var1 Sample Data	Var1 and Var2 Schema Snippet

<pre><ns1:OrderInfo
    xmlns:ns1="http://
      temp.com">
<ns1:OrderHeader>
 <ns1:OrdId>78</ns1:OrdId>
  <ns1:BillToInfo>
   <ns1:Name>Name1</ns1:Name>
   (Addr1 is missing)
   <ns1:Addr2>1 Main St
      </ns1:Addr2>
   <ns1:City>Albany</ns1:City>
   <ns1:St>NY</ns1:St>
   <ns1:Zip>12012</ns1:Zip>
   <ns1:Cntry>USA</ns1:Cntry>
  </ns1:BillToInfo>
...</pre>
	<pre><xs:complexType
	      name="AddressInfoType">
	 <xs:sequence>
	  <xs:element ref="ord:Name"/>
	  <xs:elementref="ord:Addr1"minOccurs="0"/>
	  <xs:element ref="ord:Addr2"
	       />
	  <xs:element ref="ord:City"/>
	  <xs:element ref="ord:St"/>
	  <xs:element ref="ord:Zip"/>
	  <xs:element ref="ord:Cntry"
	       />
	 </xs:sequence>
	 </xs:complexType>
	<xs:element name="BillToInfo"
	     type="ord:AddressInfoType"/></pre>

With Disable *bpws:selectionFailure Fault* and *Auto Create Target Path for Copy/To enabled*, the process terminates normally because the empty selection node is allowed, and the location path to the empty selection is built. A null value is added to `Addr1` during the assignment.

Setting Debug Preferences

You will find several pages of preference settings for the Debug perspective when you select **Window > Preferences** from the main toolbar.

Most of the settings are default Eclipse settings and are not applicable for Process Developer.

The following settings are applicable for Process Developer.

Run/Debug Preference Page	Applicable Setting
Console	All settings on this page apply to the Process Console
External Tools	not applicable
Launching	Remove terminated launches when a new launch is created
String substitution	not applicable
View Management	not applicable

Debugging Remote Processes Running on the Server

As described in Chapter 26, "Deploying Your Processes " on page 343, you can run your BPEL processes on A Process Server. You can connect to a running, suspended, completed, or faulted process from within Process Developer for remote debugging.

When the process instance is opened in Process Developer, you have full access to the debug features you use during simulation, such as setting a breakpoint, stepping or running to a breakpoint, following highlights along the execution path, and inspecting process variables. You can also modify data values and retry activities.

Before you can debug a process remotely, you must complete the following:

- Create and validate a BPEL process
- Create a process deployment descriptor file
- Create a business process archive
- Deploy the business process archive to the Process Server
- Keep a copy of the BPEL and WSDL files in your workspace project
- Start the server
- Instantiate the process by sending an appropriate message to it

Select the following topics for remote debugging details.

- "Setting Up the Embedded Process Server" on page 38
- "Configuring a Remote Process Connection " on page 329
- "Selecting Processes for Remote Debugging " on page 331
- "Setting Breakpoints on a Process for Remote Debugging " on page 332
- "Using the Debug View Process Editor and Variable View for Remote Debugging" on page 333
- "Remote Debugging Preferences " on page 334
- "Selecting a Launch Configuration from the Toolbar " on page 335
- "Checking for an Out of Sync Process " on page 335
- "Server Interactions During a Remote Debugging Session " on page 336
- "Setting Options for Console Output " on page 336
- "Correcting Retrying or Completing Activities " on page 336
- Chapter 30, "Process Exception Management" on page 420

Configuring a Remote Process Connection

Set up a configuration to indicate where processes are running and how you want to attach to them.

As a first step in remote debugging, you must set up a configuration to indicate where processes are running and how you want to attach to them.

1. From the **Run** menu, select **Debug Configurations**.

2. In the Debug Configurations dialog, select Remote Process, and then select **New**.

3. In the Name field, you can change the default configuration name, if desired, to make it more meaningful. This is useful if you set up multiple configurations to various servers or for various authenticated users.

4. In the Main tab, the default Server URL is displayed for the Process Developer engine embedded with Process Developer. Modify this URL if you want to point to the Process Developer engine in another location or to a different Process Server.

5. In the Process Selection panel, select what processes you want to debug, and refer to "Selecting Processes for Remote Debugging " on page 331 for details.

Manual process selection or breakpoints	A list of attached processes appears in the Debug view, and you can right-mouse click on the process to connect to it.
Prompt for process selection from list	When you select Debug, a dialog box appears displaying running or completed processes available for debugging
Stop on next process instance creation	When you select Debug, a "Waiting" message appears until a process instance is created for any deployed process. Once the process is instantiated, the local copy of the process opens in the Process Editor for debugging. Refer to Options Tab below to hide the waiting message for the current debug configuration and to "Remote Debugging Preferences " on page 334 to hide the waiting message for all processes.

6. In the Process location field, type in a process name, if desired. Only processes of this name are attached to during instance creation and are used as the default filter for process lists.
If desired, you can type in the qualified name, enclosing the namespace in curly braces, as shown in the example. By using the qualified name, you can pinpoint processes of similar names using different namespaces.

{http://acme.com/loanprocessing}loanApprovalProcess

7. Enable the option Attach to any process on breakpoint, if desired, to indicate that any local copy of a process containing a breakpoint is attached to, if not already attached, when a breakpoint is hit.
Do not enable this option for processes running on a high volume server. Remote debugging can impact performance.

Authentication Tab

1. Select one of the following:
   - No credentials required. No username and password are required to access the Process Server.
   - Always prompt for credentials when connecting. Prompt for username and password when beginning a remote debug session for a running process on Process Server.
   - Use supplied credentials when connecting. Use the Username and Password filled in below when connecting to the Process Server.

2. Enter a Username and Password, if required, for the Server URL.

Options Tab

You can select options for a particular debug configuration that override the default settings listed in Preferences. Preferences are located in **Window > Preferences > Process Developer > Remote Debug**. For details on the default settings, see "Remote Debugging Preferences " on page 334.

Common Tab

1. Select the Common tab.

2.   Modify the following defaults, if desired:

- Local or Shared File. You can select Shared and then save the currently named configuration to a workspace folder to share it. The configuration settings are added to an XML file named *configuration_name*.launch.

- Display in Favorites menu. By default the Debug and Run icons are displayed on the Process Developer toolbar.

- Standard Input and Output. By default, input is displayed on the Console, and you can also send output to a file.

Starting a Debug Session

Select one of the following to start or delay a remote debug session.

- Select **Apply** to save this configuration and continue creating another configuration

- If the server is running, select **Debug** to save and begin remote debugging for the current configuration

- If the server is not running, select **Close**. The configuration is saved, and you can run it from the Debug As list in the main toolbar.

Selecting Processes for Remote Debugging

Select a process by picking from the list, typing in an ID, or typing in a name. The list shows running processes. You can select All, Completed, or Faulted from the drop-down list.

When you create a configuration for attaching to remote processes, as described in "Configuring a Remote Process Connection " on page 329, you can select how you want to attach to a process.

Here are some details on each selection type.

Manual process selection or breakpoints

You can set up a launch configuration and then connect to a process to debug it. As the example shows, select **Connect to Process** from the launch configuration, or from the Debug toolbar (the launch configuration must be selected to add focus to the toolbar). The Select Process dialog displays.
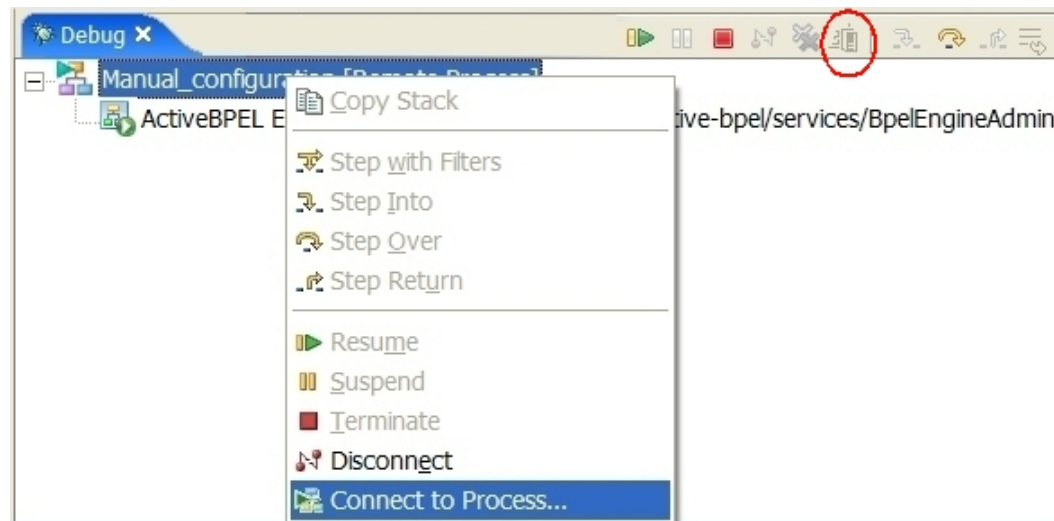
Select process by Id or from a list

When you start up debugging, by selecting **Debug** in the Debug dialog, the **Select Process** dialog appears, as shown in the example.
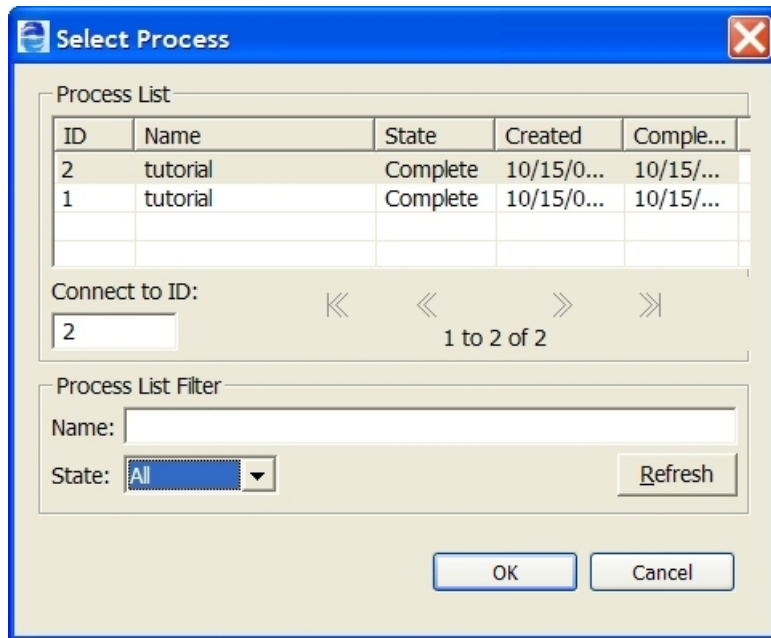
By default, the Process List displays running processes. To make other selections, do one of the following:

- Select a process from the list or type an ID in the Connect to ID box.
- Select process instances of a process by typing a name in the Name field and selecting **Refresh**.
- Select All, Running, Complete, Faulted, Suspended, or Compensatable from the **State** menu

The **Select Process** dialog can display the namespace plus the process name, if desired. For details, see "Remote Debugging Preferences " on page 334.

Stop on next process instance creation

When you start up debugging, the following dialog can appear, if you set a preference to show it.
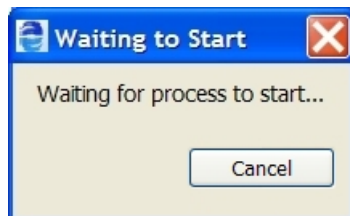
When any deployed process is instantiated, the local copy of it in your workspace is opened in the Process Editor. The process is suspended at the initial activity. You can run through or step through the process.

You can set a preference to show/hide the Waiting to **Start** dialog. For details, see "Remote Debugging Preferences " on page 334.

Setting Breakpoints on a Process for Remote Debugging

You can set a breakpoint on any process. Set the breakpoint on your workspace copy of the process. You do not need to deploy a process with breakpoints set. For more information, see "Running to a Breakpoint in a BPEL Process " on page 315.

You can attach to any running process on breakpoint by enabling an option in the Debug dialog. With this option enabled, all workspace copies of your processes are scanned to see if any breakpoints are set.

Process Developer connects to any running process when a breakpoint is hit and begins a remote debug session.

Using the Debug View Process Editor and Variable View for Remote Debugging

Using the Debug View, Process Editor, and Variable View for Remote Debugging

In remote debugging, the debug target is the remote process engine. The debug target contains multiple processes that were attached to for debugging.

When you attach to a process, it is suspended by the debug target until you resume or step the process.

In the Debug View, you can suspend, disconnect, and terminate all threads. In addition, you can use tools to make corrections in faulting activities, as shown in the illustration.
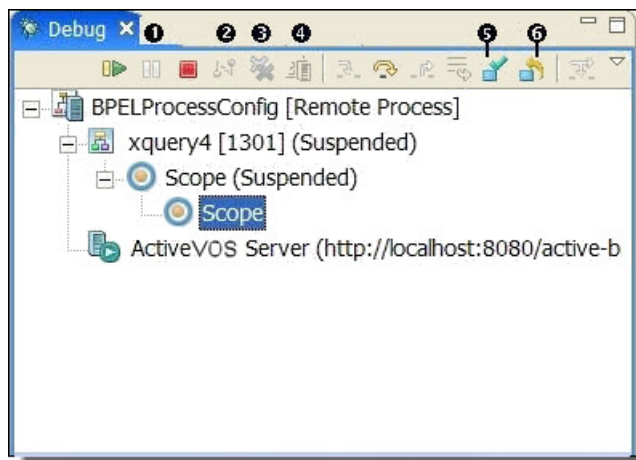
1	Suspend	Suspends the selected running process
2	Disconnect	Disconnects from the remote server resuming any processes that were suspended
3	Remove All	Removes all terminated processes
4	Connect to Process	Set up a launch configuration and then connect to a process
5	Complete Activity	Complete an activity without executing it. For details see "Correcting Retrying or Completing Activities " on page 336.
6	Retry Activity	Retry execution of an activity. For details see "Correcting Retrying or Completing Activities " on page 336.

Once a process is attached, Process Developer automatically opens the workspace copy of it in the Process Editor. The Process Editor shows the state of the attached process instance, in the same way as it does for simulation as the example shows.
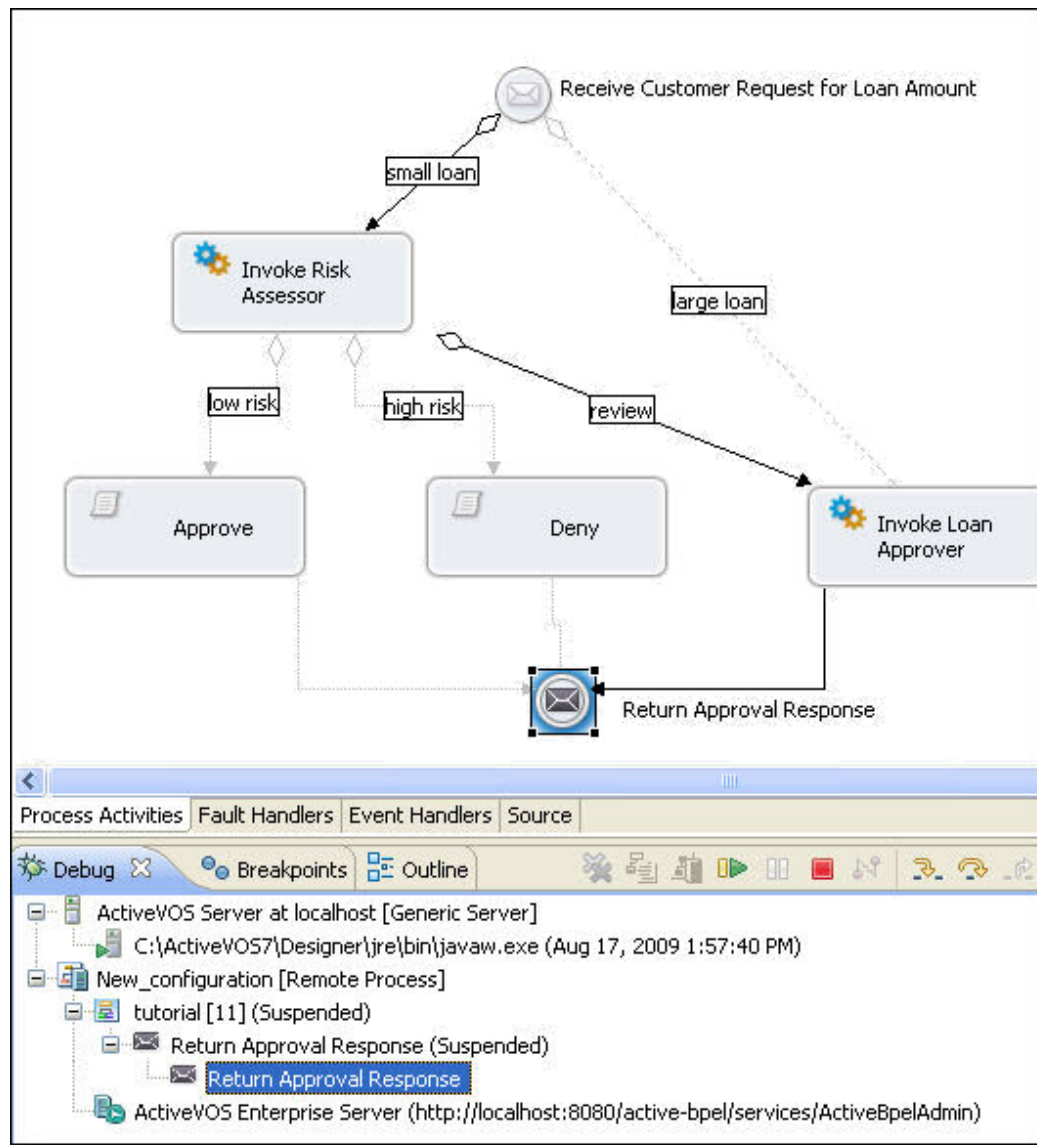
The Process Variables view is automatically updated to follow the instance data of the selected process.

Remote Debugging Preferences

Set remote debugging preferences for process selection.

You can set remote debugging preferences as follows:

1. From the Window menu, select **Preferences**.

2. Select **Process Developer > Remote Debug**.

3. Set preferences for:

    -   **Process List Limit**. Set the number of processes to view in the Select Process dialog. This dialog appears when you want to be prompted for a process to debug.

- **Show qualified process names**. Enable this setting to view the process name qualified by a namespace. The process name is displayed in the Select Process dialog. If you have two similar process names from different namespaces, this option helps you select the correct process.

- **Show Waiting to Start dialog**. When you want to debug the next process instance that occurs, the **Waiting for Process** dialog appears. Enable this setting to show the dialog. When the next process is instantiated, the local copy of the process appears in the Process Editor and is suspended in the Debug view.

- **Disconnect launch when last connected process terminates**. Enable this checkbox to automatically disconnect the launch and server when the last executing process terminates. If you leave the checkbox blank, you can select the Disconnect icon on the Debug view toolbar for each launch.

- **Check for out of sync process definition when connecting**. Enable this setting to check that the BPEL process definition on the server matches the BPEL process definition in your Project Explorer view. The process you want to debug remotely can be an older or newer version than the process in your workspace. During remote debugging, Process Developer uses your workspace copy of the process and the actual instance data from the server to perform debugging. This means the workspace copy of the process should be identical to what is running on the server. It is recommended that you leave this option checked. If a process is out of sync, Process Developer displays an error message during remote debugging.

- **On disconnect of suspended process**. Select whether you want to be prompted, resume running the process, or take no action.

Selecting a Launch Configuration from the Toolbar

You can select a named debug configuration from the toolbar. You can use this shortcut to start a remote debugging session. The Debug picklist shows connection configurations, as the example shows.
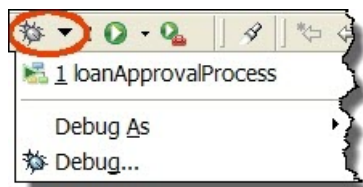
See "Configuring a Remote Process Connection " on page 329 for more information.

Checking for an Out of Sync Process

If this invoke activity is defined with a BPEL process, you can step into it during simulation. Select a different process to synchronize with the process deployed to the server.

You can set a Remote Debug preference to check for an out of sync process definition. This means that Process Developer checks that your local copy of the process matches the server copy when connecting to the server.

During remote debugging, Process Developer uses your workspace copy of the process and the actual instance data from the server to perform debugging. The workspace copy of the process should be identical to what is running on the server.

However, if the process definitions do not match, you can perform remote debugging anyway, and Process Developer does its best to step through the activities.

With the preference enabled, if the server copy and the local copy of your process are out of sync, Process Developer give you an option to choose another process, cancel, or continue to starting remote debugging, as the illustration shows.
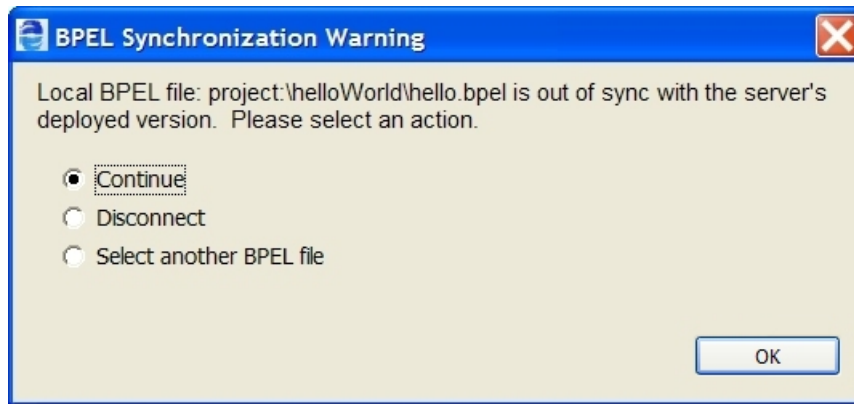
You can ignore the warning and continue to debug or select another process definition.

If you disable the out of sync option, remote debugging begins, but an error messages is displayed when the mismatch is reached.

Server Interactions During a Remote Debugging Session

During remote debugging, the following occurs:

- When you suspend a process, the process is suspended on the server
- When you step the process, the process resumes on the server
- When you terminate the process, the process terminates on the server
- When you change data values, they are changed on the server
- The Console shows processing steps

For more information about Console output, see "Setting Options for Console Output " on page 336.

Setting Options for Console Output

When a remote debug session attaches to a process manually, or by connecting on a breakpoint, some event logging for that process may have already taken place in the Process Server. If so, that log is sent to the remote debug target and displayed in the Console view. If logging is turned off in the engine, no log exists, and the message "Preceding events for this process are not available" appears, followed by individual events, logged as they are received by the debugger from the engine.

For more information regarding setting logging options, refer to the Process Server documentation.

Correcting Retrying or Completing Activities

You can make changes to a simulated or running process and then use the Debug view toolbar actions to retry or complete activities.

The following illustration shows the Debug toolbar actions.
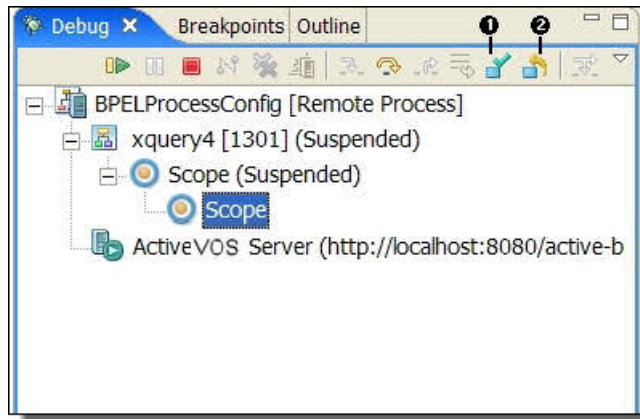
| 1 | Complete the activity in focus, effectively stepping over it without executing it. |
| 2 | Retry the execution of the activity in focus . |

See Also:

- "Updating Variable Data in the Process Variables View " on page 338
- "Updating Correlation Property Data " on page 340

Updating Variable Data in the Process Variables View

When you are remote debugging a process, you can update the values in your variable data. This allows you to fix data that may be causing faults or undesired execution. You update a variable in the same way as in simulation.

1. Begin a remote debugging session. For details, see "Debugging Remote Processes Running on the Server" on page 329.

   Begin a remote debugging session. For details, see "Debugging Remote Processes Running on the Server" on page 329.

   The following illustration shows an example of a process that receives an invalid message (missing `<data2>` element). The assign activity is faulting as a result.
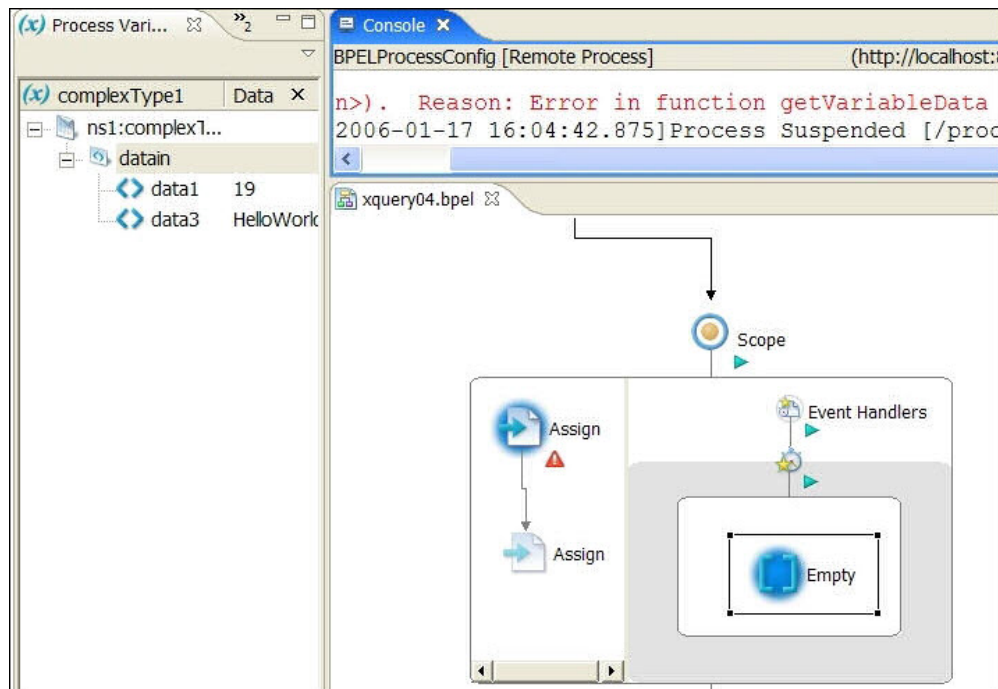
2. Correct a data value by editing a simple type or by loading a valid sample data file for a complex type. Right-mouse click on the message part and select **Edit Data** or **Load Data**. The following example shows a complex message for which you can load a sample data file.
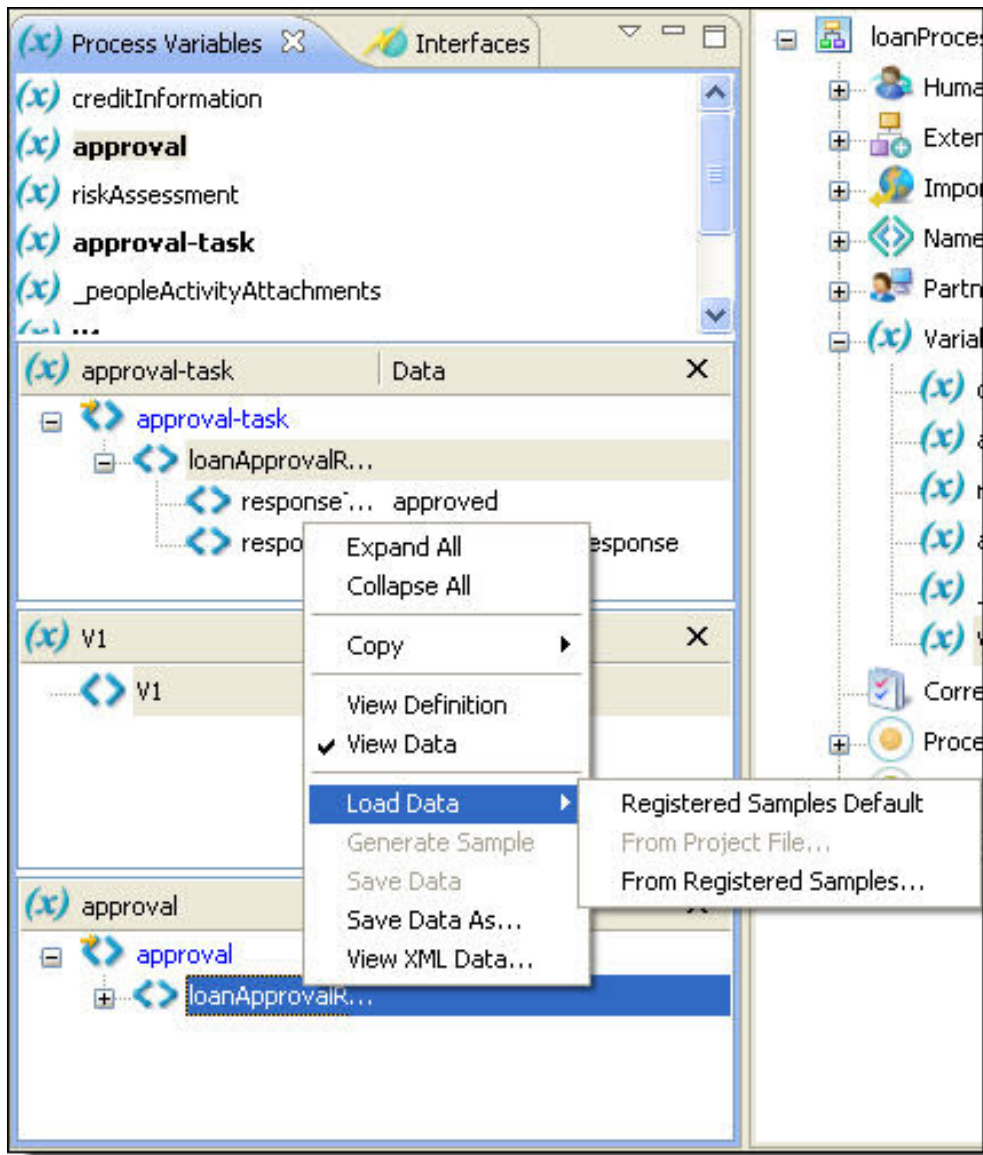
3. After you have updated the value of the variable, you can retry an activity by selecting the **Retry** button in the Debug view.

Updating Correlation Property Data

If the Console indicates a correlation violation, you can update the value of the correlation proprieties.

To update correlation properties:

1. Begin a simulation or remote debugging session. For details, see "Debugging Remote Processes Running on the Server " on page 329.

    The following image shows a faulting activity caused by a correlation violation, where the correlation property of the Request and Receive variables does not have a matching value:
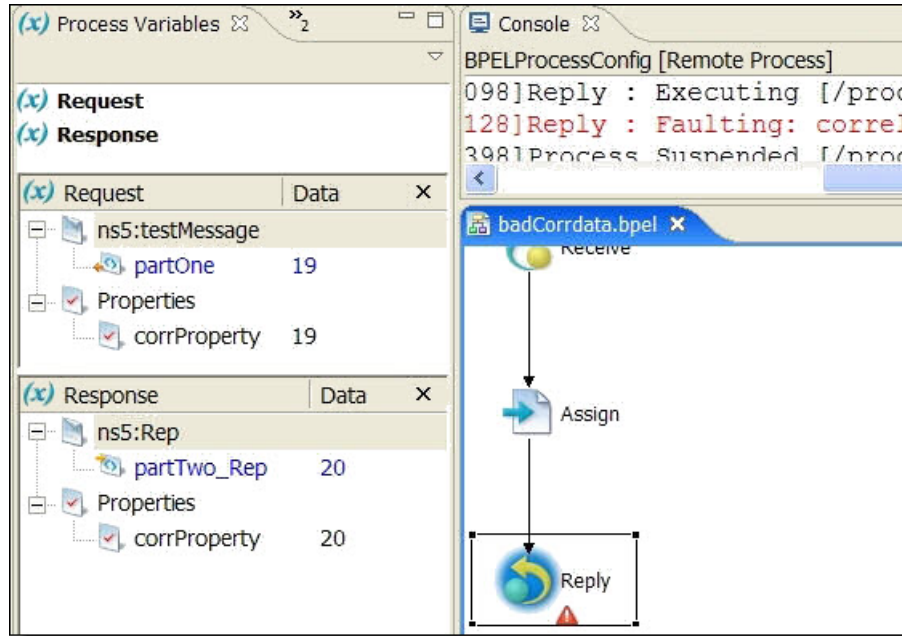
2. To correct the correlation violation, navigate to the Outline view and select the correlation set.

3. In the Properties view, under the Execution category, select the **Correlation Set Data** dialog button. The example shows the dialog button at the end of the row. Note that the Execution property is only available during simulation or if you are using remote debugging. It does not normally appear in the Properties view.
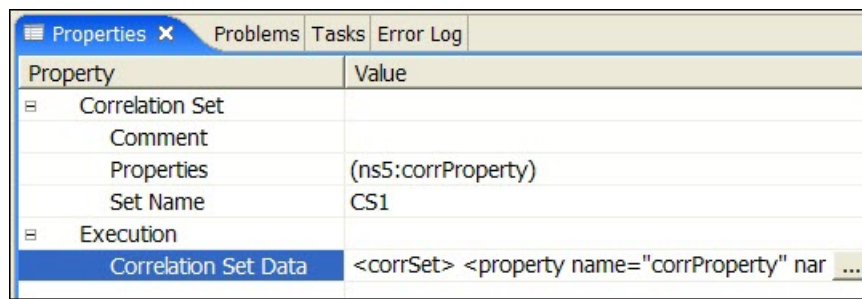
4. Type in the expected value for the correlation property. The following image shows an example.

5. Select **OK**, and navigate to the Debug view.

6. You can retry an activity by selecting the **Retry** button on the Debug view.

Updating Partner Link Address Information

During simulation or debugging, you may need to modify the partner link address, which may be specified in the Process Deployment Descriptor (PDD) file or dynamically assign in the process.

By updating the partner link execution state data, you enable copying endpoint reference information into other variables. For example, you can be working with WS Addressing properties in message headers that you are copying to a process variable. You can set this type of information in a partner link address during simulation or in a remote debugging session.

1. Start a simulation or remote debug session.

2. Navigate to the Outline view and select a partner link.

3. In the Properties view, under the Simulation category, select the My Role or Partner Role partner link, as needed. The following illustration shows an example.
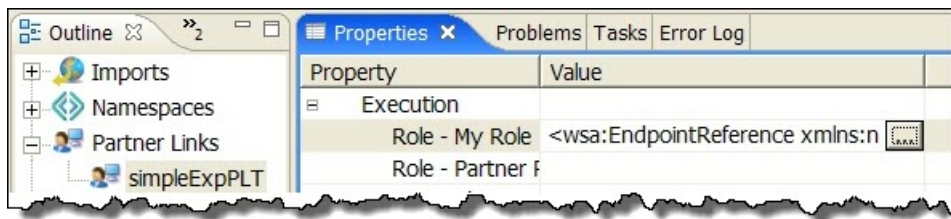
4. Select the Dialog button at the end of the row to open the Partner Link Execution State dialog, as the example shows.
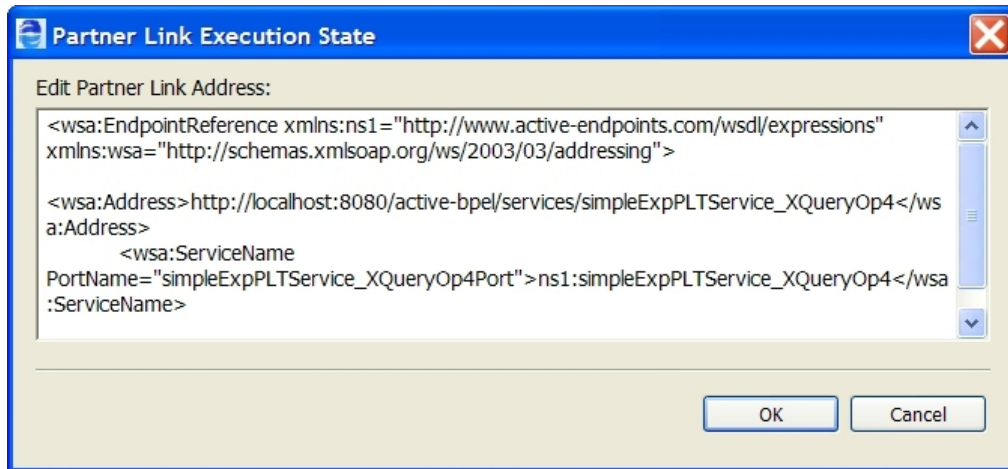
5. Modify the address as needed.

6. If you are remote debugging (on-premises only), you can retry an activity associated with an endpoint reference by selecting the **Retry** button on the Debug toolbar.

Monitoring Client Message Traffic with TCP IP Monitor

The Eclipse Web Tools Project, included with Process Developer, contains a view that allows you to monitor SOAP messages as they are sent and received from custom client applications your process is communicating with. Process Developer includes this project and view for your testing convenience. Follow the instructions in the *Web Tools Platform User Guide* to configure the TCP/IP Monitor.

CHAPTER 26

Deploying Your Processes

Deployment is the act of publishing your BPEL process to A Process Server where it can run.

In Process Developer, you create the BPEL process and associated resources required for the Process Server. You then create a process deployment descriptor. You package the deployment descriptor and all associated resources in a business process archive and export it to the server.

When your process runs, the Process Server ensures that the correct partner services and messages interact with your process.

The Process Server uses the WS Addressing protocol to identify the partner endpoints that the process communicates with.

The following topics discuss deploying your processes.

- "Preparing for Deployment " on page 343
- "Endpoint Reference Addressing Considerations " on page 344
- "Overview of Process Deployment Steps " on page 346
- "Creating a Process Deployment Descriptor File " on page 347
- "Creating and Deploying a Business Process Archive Contribution " on page 372
- "Managing Deployment Contributions " on page 377
- "Using a BPRD Script to Regenerate and Deploy a BPR Contribution " on page 378
- "How a BPEL Process is Instantiated " on page 381
- "What is Process Versioning " on page 381

Preparing for Deployment

Preparing your BPEL process for deployment to the Process Server requires two tasks:

- "Preparing BPEL Files for Deployment " on page 344
- "Selecting a Server Platform for Deployed Processes " on page 344

In addition, you may wish to read "Endpoint Reference Addressing Considerations " on page 344 before creating the process deployment descriptor file. This topic describes the details you may need for providing necessary deployment information.

If your partner's service requires authenticated access, see "Endpoint References Requiring Credentials for Access " on page 345.

Preparing BPEL Files for Deployment

A BPEL file is ready for deployment to the Process Server when there are no errors in the Problems view and when a simulated execution of the process terminates normally.

To ensure your process is deployment-ready, complete this checklist:

- Ensure that you have filled in all the required properties for each activity, such as partner link, operation and variable for receives (picks and event handlers), invokes, and replies.

- Ensure that at least one receive or pick activity's Create Instance property is set to Yes.

- Ensure that the Abstract Process property is set to No for the process.

- Step through a simulated execution of the process, as described in Chapter 25, "Simulating and Debugging " on page 306.

Selecting a Server Platform for Deployed Processes

Your BPEL process can run on any BPEL server. However, to deploy it to A Process Server, make a package of files (`.bpel`, `.wsdl`, `.pdd`) for deployment as described in "Creating and Deploying a Business Process Archive Contribution " on page 372.

Process Developer includes the Process Server in your installation folder. You can deploy processes to this development server as you design and test processes. Instructions for deploying to this server are included in "Creating and Deploying a Business Process Archive Contribution " on page 372.

Endpoint Reference Addressing Considerations

Process Server supports a variety of addressing options to specify how, when, and where to reach the Web services invoked in a BPEL process. During the creation of the process deployment descriptor (PDD) file, you can add many address details.

Endpoint References and WS-Addressing Considerations

At deployment time, you need to specify the location information for each partner link endpoint reference. Also, you can assign an endpoint reference dynamically, detailed in the Copy Operation Dynamic Endpoint Reference Example.

One of the main protocols to specify location information for an endpoint reference is based on the Web Services Addressing (WS-Addressing) specification.

The WS-Addressing specification standardizes the format for referencing a Web service allowing you to create uniquely identified Web services and Web service instances that are not bound to a specific transport mechanism, such as HTTP. Using WS-Addressing, the endpoint information is added to the header of a SOAP request and not to the URL specified in the SOAP body.

Having endpoint reference information in a SOAP header means that you can do the following:

- Specify Web service instance IDs

- Select *Reply To* and *Fault To* addressees

- Select *From* endpoint references in the case where an acknowledgement needs to be sent back to the sender

Note that in Process Server, you can specify other mechanisms besides SOAP over HTTP. For details, see *Custom Service Interactions*.

The basic WS-Addressing syntax is as follows:

```
<wsa:EndpointReference>
  <wsa:Address>anyURI</wsa:Address>
  <wsa:ServiceName PortName="portname">Service QName
    </wsa:ServiceName>
</wsa:EndpointReference>
```

where:

- `Address` is a mandatory element that identifies an endpoint. It can be a network address or a logical address.

- `ServiceName` is the qualified name, or QName, for the service being invoked. This service must be defined in a WSDL file that is deployed with the process or one that is available in Process Server's resource catalog.

- `PortName` identifies the service port being invoked. Note that you can specify a port binding of SOAP 1.1 or SOAP 1.2.

You can add credential details to the endpoint reference, described in *Endpoint References Requiring Credentials for Access*.

Note: For information on setting and endpoint reference when deploying to Informatica Cloud, see .

The following versions of the WS-Addressing specification are supported:

- `http://www.w3.org/2005/08/addressing`

  This is the default version that the PDD uses. For changing the default, see *Process Developer Preferences*.

- `http://schemas.xmlsoap.org/ws/2004/03/addressing`

- `http://schemas.xmlsoap.org/ws/2004/08/addressing`

- `http://schemas.xmlsoap.org/ws/2003/03/addressing`

You can review the WS-Addressing specification at `http://www.w3.org/2005/08/addressing/`.

Endpoint References Requiring Credentials for Access

At deployment time, you must specify the location information for each partner link endpoint reference. If a partner's service requires authentication for access, you can add credential details to the partner link definition in the process deployment descriptor file. For details, see "Adding Policy Assertions " on page 357.

Specifying a Replaceable URN URL for an Endpoint Reference

You can specify a logical or physical address for a static endpoint reference in the PDD file. If you specify a logical address, or URN, you can then map the URN to the physical address in the URN Mappings page of the Process Console. If you specify a URL for the static endpoint address, you can replace the URL by mapping it to a different URL on the server.

For example, in the PDD editor, you can specify the following address:

```
<wsa:Address>urn:localhost:AssessRisk</wsa:Address>
```

In the Process Console, after you deploy the process, you can map the URN to a URL as follows:

```
urn:localhost:AssessRisk = http://localhost:8080/active-bpel/services/AssessRisk
```

For more examples of URN syntax, see the *URN Mappings* topic in the *Process Console Help*.

Endpoint References and WS-Policy

The Process Server supports the Web Services Policy Framework (WS-Policy). A policy can describe a broad range of service requirements, preferences, and capabilities. You can add policy information to the endpoint reference section of a process deployment descriptor file. For details, see *Adding Policy Assertions*.

Overview of Process Deployment Steps

The following illustration shows how you can take your BPEL process from design to deployment.
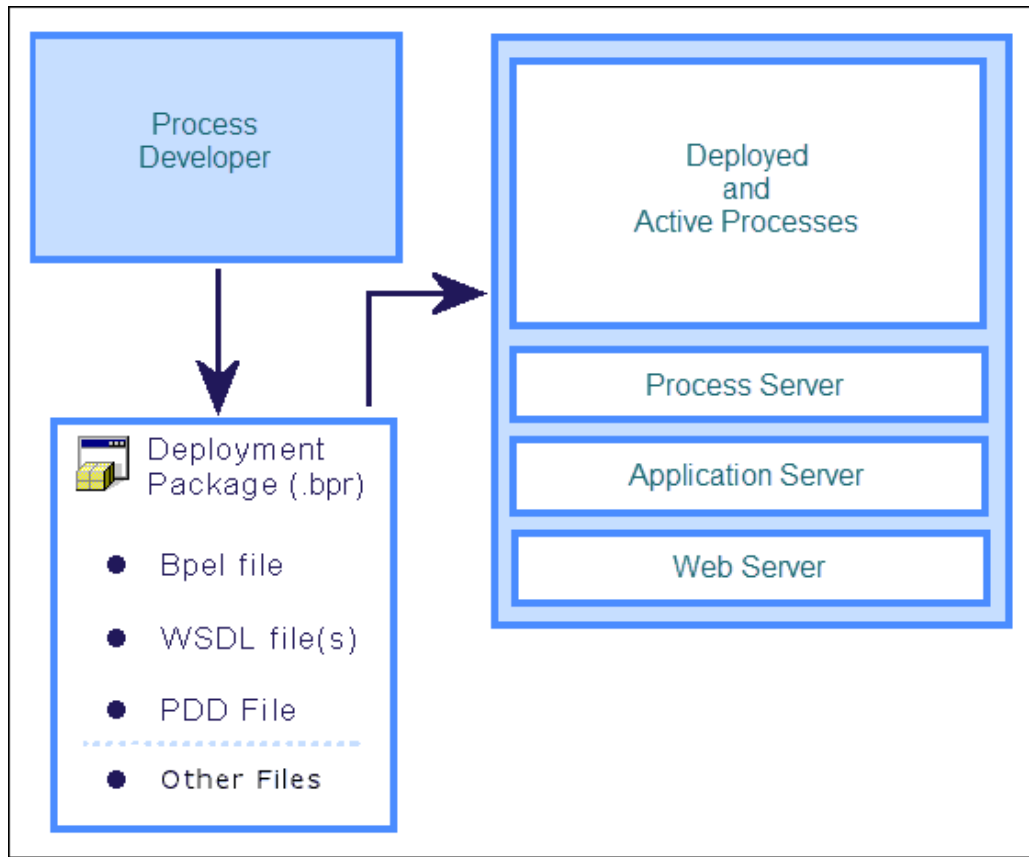
1. In Process Developer, import WSDL or service references to your project as needed, create the BPEL file and the Process Deployment Descriptor (`.pdd`) file. See "Creating a Process Deployment Descriptor File " on page 347 for details.

2. In Process Developer, create and export the process deployment package (`.bpr`) file. See "Creating and Deploying a Business Process Archive Contribution " on page 372 for details.

Once the process is deployed, it becomes active when an initial receive activity (or pick or event) receives a message.

Creating a Process Deployment Descriptor File

A Process Deployment Descriptor (.pdd) file describes the information required for a process to execute in the Process Server environment. If you are deploying a process to a production server, you can specify process version and other details to override the server defaults.

A Process Deployment Descriptor (.ppd) file describes the information required for a process to execute in the Process Server. This information includes partner link details, persistence and versioning information, process directives and indexed properties as well as other details. In each step of the procedure below, there are links to the relevant help topics for the deployment details to add.

To create the process deployment descriptor file:

1. In Process Developer, select **File > New > Deployment Descriptor**.

2. Select a folder and a process deployment descriptor filename, if you want to change the defaults.

3. Select a BPEL file to be described, and select Finish to open the PDD Editor.

4. Select General options. For details see *General Deployment Options*.

5. Fill in the Partner Links tab for each partner link as follows.

    a. Select a partner link from the list.

    b. If the partner link has a Partner Role defined, select the Endpoint Type for the partner link, and optionally, select an Invoke Handler. For explanations and examples of Endpoint Types, see *Partner Role Endpoint Types*. For explanations of Invoke Handlers, see *Partner Role Invoke Handlers*. For WS-Policy details, see *Endpoint References and WS-Policy*.

    c. If the Endpoint Type is Static, see the notes below.

    d. If the partner link has a My Role defined, select the Binding style and Service Name. Optionally, you can specify Allowed Roles. For explanations of these properties, see *My Role Binding, Service Name, and Allowed Roles Options*. To add WS Policy assertions to a My Role partner link, see *Adding Policy Assertions*.

6. For Process Server, define Indexed Properties, if desired. An indexed property is a variable property that is populated with the variable (or variable part) value during process execution. In the Process Console, you can select indexed properties as a search filter for process instances. In addition, you can use indexed properties for the event tracking. For details, see *Adding Indexed Properties*.

For details on adding Human Tasks Logical People Groups, refer to Human Tasks elsewhere in this help.

Notes for Static Endpoint Type:

- For flexibility, you can specify the address as a URN rather than a URL, supplying WSA Address for the Invoke Handler. For details, see *Specifying a Replaceable URN/URL for an Endpoint Reference*.

- If the WSDL does not include bindings, WS Addressing elements are added with placeholders, as the following example shows. You can edit the placeholders on the Partner Links tab. For more information, see *Using the PDD Editor Source View*.

```
Endpoint Reference:

<wsa:EndpointReference xmlns:s='FILL_IN_NAMESPACE'>
  <wsa:Address>s:FILL_IN_ADDRESS_URI</wsa:Address>
  <wsa:ServiceName PortName='FILL_IN_PORT_NAME'>s:FILL_IN_SERVI
</wsa:EndpointReference>
```

- If you want to select a different service, you can. See *Selecting a Service for a Deployment Descriptor Partner Link* for details.

- To add WS-Policy policy assertions, see *Adding Policy Assertions*.

General Deployment Options

If you are deploying a process to a production server, you can specify process version and other details to override the server defaults.

During the creation of the Process Deployment Descriptor file, you can select deployment options for new and running versions. If you do not change any settings, the server uses the defaults, as described below.

Process Logging

By default the Process Server generates an execution log for running processes. You can enable this setting to override the engine default, and then view or download an execution log for a running or completed process. An execution log provides start/end times for activity execution and other details, and helps you troubleshoot faulted processes. The logging levels are:

- System Default. Setting that is enabled in the Process Console that applies to all processes without a Process Logging override. The server is set to Execution by default.
- None. No logging occurs as processes run. Performance is enhanced with logging disabled.
- Full. All execution statements are logged, including the *Will Not Execute* statements for deadpath activities. For example, all fault handling statements that are not executed are logged.
- Execution (default). All execution statements are logged, except for *Will Not Execute* statements. Using this setting instead of Full can greatly decrease the size of the log file.
- Execution with Data. All execution statements are logged, except for *Will Not Execute* statements, including variable, expression, and partner link data. Using this setting can greatly increase the size of the log file.

Process Persistence

Persistence refers to storage of active processes deployed to a server. When a process runs on the Process Server, all state and variable data is stored in the server database by default. However, for performance reasons and database size considerations, you may want to set a lower level of persistence.

If your process uses a wait activity or a retry policy, you must select the Persist or Full level.

You can make persistence setting selections as follows. Your selection is validated when you save the PDD.

Setting	Description	Restrictions
Full	The default setting. For each process instance, all state information is stored for a running, faulted, and completed process. In the event of a server failure, a running process can be fully recovered. The recovery is possible because Process Server maintains journaling (a journal of the changes intended for the database) for this setting.	If the process uses a WS-RM invoke handler for a partner role or a WS-Reliable Messaging policy assertion on a my role, you must select this setting. For details, see "Partner Role Invoke Handlers " on page 351 and "WS-Reliable Messaging " on page 369.
Persist	Same storage setting as Full, but without journaling. A running process is suspended. The process is recoverable if the system goes down, but needs to be looked at since no journaling was done, so recovery marks this type as suspended. This is the minimum persistence level to support Suspend on Uncaught Fault (see "Suspending a Process on Uncaught Faults " on page 420).	Cannot select this setting if the process uses WS-Reliable Messaging. If you reach the Activity Execution Limit set in the Tenant Detail and persistence is not set to this minimum level, the Suspend on Uncaught Fault flag is ignored and the process is terminated.

Setting	Description	Restrictions
Final	Stores only the final state of the process (completed or faulted) and process variables. On a server failure, a running process is terminated. This setting makes fewer database writes than the settings above, but allows you to view a graph of the process on the Active Processes detail page in the Process Console, where you can see the execution path and final values of process variables. A process runs only in memory, and the server property called Process Idle Timeout has no effect on this persistence level.	For Final, Brief, and None, the process cannot contain the following constructs: - Wait activity - Multiple receive activities - People activity (on-premises only) - Process event handler - Process:Subprocess invoke handler - WS-RM invokes
Brief	This is the minimum level for process logging (described in the section above), but does not allow for viewing a graph of the active process. Stores only the start and completion times as well as final state (completed or faulted). Stores state and process variables only if the process faults. A process runs only in memory, and the server property called Process Idle Timeout has no effect on this persistence level.	See Final.
None	No process information is stored in the server database when a process terminates. The process instance is not listed in the Process Console's Active Processes page.	See Final.

Group

You can assign a group name to one or more processes. The group name is a selection filter for viewing a list of deployed or active processes in the Process Console.

Suspend on Uncaught Fault

According to the WS-BPEL 2.0 specification, a process with an uncaught fault must exit. However, you can suspend this process on an uncaught fault to perform process exception management, if desired.

You can make selections as follows:

System Default	The current engine setting for all processes. The default engine setting is to disable suspension on uncaught fault.
False	Do not allow this process to suspend on an uncaught fault. The process will terminate abnormally. This setting overrides the engine setting.
True	Suspend this process on an uncaught fault to put it in a suspended-faulting state. You can then perform process exception management on the faulting process, followed by retrying or completing the faulting activity or scope. This setting overrides the engine setting. For details, see Chapter 30, "Process Exception Management" on page 420.

Suspend on Invoke Recovery

For invoke activities that do not complete due to the node failure, you can suspend the process upon recovery. The process is suspended at the pending invoke, and you can perform process exception management, is desired.

You can make selections as follows:

System Default	The current engine setting for all processes. The default engine setting is to disable suspension upon process recovery when there is a pending invoke.
False	Do not allow this process to suspend at a pending invoke after process recovery. The process will terminate abnormally.
True	Suspend this process on recovery when there is a pending invoke to put it in a suspended state. You can then perform process exception management on the process, followed by retrying or completing the invoke. This setting overrides the engine setting.

Process Instance Retention

Once you deploy a process to the Process Server, and begin to execute it, the completed and faulted process instances are stored in the Informatica Business Process Developer database. Process Server administrators can delete old processes from the database on an automatic schedule, based on the number of days, hours or minutes you want to retain old processes before deleting them.

You can specify how many days (or hours or minutes) to retain a completed or faulted process in the Informatica Business Process Developer database by adding a retention number in the PDD. This number can be modified for the process in the Process Console.

If you don't specify a value for retention, the default engine value is applied to the process. This value is relevant only if automated database maintenance is enabled.

Versioning

Note that only one version of a process can create new process instances.

| Process Version | By default, the Process Server auto-increments the version number for new versions. Auto-incrementing is based dropping the minor version number and incrementing the major version number by one. For example, 1.5 becomes 2.0.

To define your own version number for this process, type in a number, such as 1.5. |
|---|---|
| Online Date | If the online (effective) date is blank or in the past, then the process immediately becomes the online (current) version. The online version for a given process is the one capable of creating new process instances. If the date is in the future, the server makes this version the online version when the effective date arrives.

If you intend to deploy your process to a server in a different time zone, be sure to edit the deployment descriptor file to include a time zone expression. For details, see "What is Process Versioning " on page 381. |

Offline Date	If the offline (expiration) date is blank, then the process will not go offline.
	Providing an offline date is useful if you want the process to run for a limited period of time. An offline process version cannot create new process instances, but running process instances complete normally.
	You can specify an offline date. If you intend to deploy your process to a server in a different time zone, be sure to edit the deployment descriptor file to include a time zone expression. For details, see "What is Process Versioning " on page 381.
Running Process Disposition	By default, the all process instances are maintained. The default value of Maintain Version allows process instances created under previous versions to run to completion when this version becomes online.
	You can select Migrate Version to convert process instances running against an earlier version to use the new version. Be sure to check the Server Log for warnings that are generated if an incompatibility occurs.
	Select Terminate to terminate processes running under previous versions on the offline date of the new version, regardless of whether or not the process instances are complete.
	For details, see "What is Process Versioning " on page 381

If you do not change the defaults, the `.pdd` file does not include any version details. To add or change version details, see "Using the PDD Editor Source View " on page 371.

Partner Role Invoke Handlers

A Process Deployment Descriptor (.pdd) file describes the information required for a process to execute in the Process Server environment.

By default, the Process Server invokes an endpoint by looking up the service name in the WSDL to find the endpoint (for example, the <soap:address>) and using the standard engine invocation framework. You can select a different invocation method by making one of the following selections:

Invoke handler	Invokes an endpoint based on:
WSA Address	URL in the <wsa.Address> field of the PDD file. For details, see *Selecting a Service for a Deployment Descriptor Partner Link*. Note that you can provide a URN and then map the URN to a URL in the Process Console. For details, see Specifying a Replaceable URN/URL for an Endpoint Reference.
WSDL Service Port	URL in the Service's port section of the WSDL file. Select this to use the service address as is; that is, the engine only looks at the WSDL's service port at runtime. No changes are allowed to the service address. See Selecting a Service for a Deployment Descriptor Partner Link.
WSRM:Address	Same as address above, except that the messages are sent using the WS-Reliable Messaging protocol. Select this handler if you will attach the reliable messaging policy assertion, described in WS-Reliable Messaging. An addressable ReplyTo endpoint is required.
EJB Service (*on-premises only*)	Deployed JARs, WSDL, and other resources created from an EJB Interface. Note that an EJB Service cannot be used on Apache Tomcat. It is supported on other application servers. For details, see Creating a Java Interface.
Java Service	Deployed JARs, WSDL, and other resources created from a Java Interface. For details, see Creating a Java Interface.

Invoke handler	Invokes an endpoint based on:
JMS Service	For details, see Using a Java Messaging Service Invoke Handler. Note that an addressable ReplyTo endpoint is required.
REST Service	For details, see Using a REST-based Service.
System Service	Partner role based on one of the System Services, such as Server Log Service, Email Service, *Identity Service*, *Reporting Service*, *Shell Command Invoke Service*, *Data Access Service* and *Using a REST-based Service*.
Process	BPEL process and PDD located in a workspace project and deployed to the same server as the calling process. For details, see *Creating a BPEL Process as a Service for Another BPEL Process*. To select a process, see *Select Process Service*. **Note:** Select this option when deploying to Informatica Cloud.
Process:Subprocess	BPEL process and PDD located in a workspace project and deployed to the same server as the calling process. The process is a *subprocess*; that is, it is eligible for compensation within the enclosing scope of the calling process. For details, see *Creating a BPEL Process as a Service for Another BPEL Process*. To select a process, see *Select Process Service*.
EJB Wrapper	Custom EJB you have written, packaged and deployed to A Process Server that supports EJBs. You can specify the JNDI location of the EJB as well as the query string required to invoke the endpoint. For details, see *EJB/Java Invoke Handler Properties Dialog*.
Java Wrapper	Custom Java class you have written, packaged and deployed to the server. You can specify the fully qualified class name for the Java class as well as the query string required to invoke the endpoint. For details, see *EJB/Java Invoke Handler Properties Dialog*.

Partner Role Endpoint Types

A Process Deployment Descriptor (.pdd) file describes the information required for a process to execute in the Process Server environment. When creating the .pdd file, you must select an endpoint type for each partner role defined in a partner link.

During the creation of the Process Deployment Descriptor file, you must select an endpoint type for each partner role defined in a partner link. An *endpoint type* is a binding property that indicates how to communicate with the Web service the process interacts with. The different types give you control over specifying services you work with now and in the future. For example, you can "hard-code" a service binding by making it static, or you can select more versatile options for changing and adding new service partners dynamically without redeploying the process.

The endpoint types are defined in the following table.

Endpoint Type	Description
Dynamic	Indicates that the service location (that is, the endpoint reference) is provided dynamically within the BPEL process. The location must be assigned dynamically within a Copy Operation of an Assign activity.
Principal	This selection is deprecated as of Version 9. The selection remains in place for legacy processes that use a partner definition file. Indicates the endpoint reference is determined based on the value of the authenticated principal that sends the request to the BPEL process. In this configuration, it is assumed that service endpoints for the BPEL process are secured. When a message arrives on one of these endpoints, the authenticated user's principal (typically their username) is used as a lookup in Process Server. The server determines the proper endpoint to use based on the principal and the partner link type being invoked. The principal endpoint type provides the security of knowing whom you are dealing with and the flexibility of being able to configure the endpoints independently from the process and its deployment. The use of principal implies an asynchronous conversation since it is used to populate the partner role used for a callback. The pattern is a Receive-Invoke where the partner link is the same for both activities. The process declares both the myRole and partnerRole attributes for the partner link. The receipt of the data for the Receive activity result in the partner link's partner endpoint reference being initialized with the callback from the partner definition file.
Static	Indicates the endpoint information is defined in the deployment descriptor file and is used for all invocations of the partner link. **Note:** Select this option when deploying to Informatica Cloud.

If the partner service requires authenticated access, see "Endpoint References Requiring Credentials for Access " on page 345.

My Role Binding Service Name and Allowed Roles Options

A Process Deployment Descriptor (.pdd) file describes the information required for a process to execute in the ActiveVOS server environment. When creating the .pdd file, you must select a binding style and service name for each service partner link of my_role type.

During the creation of the Process Deployment Descriptor file, you must select a binding style and service name for each service partner link of my_role type.

The binding styles are the standard SOAP styles of RPC (remote procedure call) and Document, and there are custom styles called Unpublished and Policy Driven.

Select the style that is appropriate for the simple, schema, or complex message variable manipulated by your service's operation:

- A **Document Literal** invocation means the entire SOAP message consists simply of a single entity that is an XML document. This invocation is equivalent to a binding style of document/literal.

- An **RPC Encoded** invocation means that the body of a SOAP request has an outer element that matches the operation name and contains inner elements each of which maps to a parameter of the operation.

- An **RPC Literal** invocation means that the body of a SOAP request has an outer element that matches the operation name and each inner part points to a schema type definition that describes the content of that part. RPC is the binding style and *literal* is the *use*.

- An **Unpublished** invocation bypasses the standard invocation framework. The primary reason for deploying with the unpublished binding is to take advantage of facilities offered by the application server with regard to securing or managing the Web service. Instead of using the built-in mechanism provided by

Process Developer for service publication, you can control the publication of your services through an invocation such as J2EE for Web Services (WS4EE). Also, Process Developer system services are specified as unpublished.

- A **Policy Driven** invocation means that the specific facilities used for handling SOAP requests are determined at deployment by the policy assertions attached to the `myRole` endpoint. For example, services that support WS-Reliable Messaging protocol can select Policy Driven as the binding style and include an WS-Reliable Messaging policy. At deployment, if the service cannot determine the binding style from the policy assertions, an error is thrown.

- The **Service Name** is the name of the endpoint that Process Server creates for the BPEL process. This name can match the address in the binding section of the WSDL file, or any name desired. By default, the service name is based on the port type associated the partner link.

- **Allowed Roles** is optional and specifies one or more security roles that an authenticated user must have in order to communicate with the BPEL process. If the application server is not configured with security. the allowed roles field is ignored. Refer to your application server documentation on how to secure individual web applications.

  Note: You must complete this field when deploying a process to Informatica Cloud. if you do not specify at least one role, no one will be able to access the process.

You can also add policy assertions to the my role partner link, as described in .

Selecting a Service for a Deployment Descriptor Partner Link

Select a service containing the binding information for the endpoint reference. Select the Policy Assertions tab to add one or more policy assertions for the endpoint reference.

Selecting a Service for a Deployment Descriptor Partner Link

During the creation of the Process Deployment Descriptor file, you must select an endpoint type for each partner role defined in a partner link. If you select a static type, to "hard-code" a service binding, you can select the service to use.

By default, the binding information from the imported WSDL is used, if only one matching service binding exists. However, if you want to bind to a different service, click **(…)** to choose another service.

The WSDL you select must be in the workspace registry and must contain binding information.

In the Endpoint Reference dialog that appears, do one of the following:

- Select a WSDL from the picklist. When a matching service appears, select it to fill in the namespace, address, service, and port in the text boxes above. The following example shows a selected service:
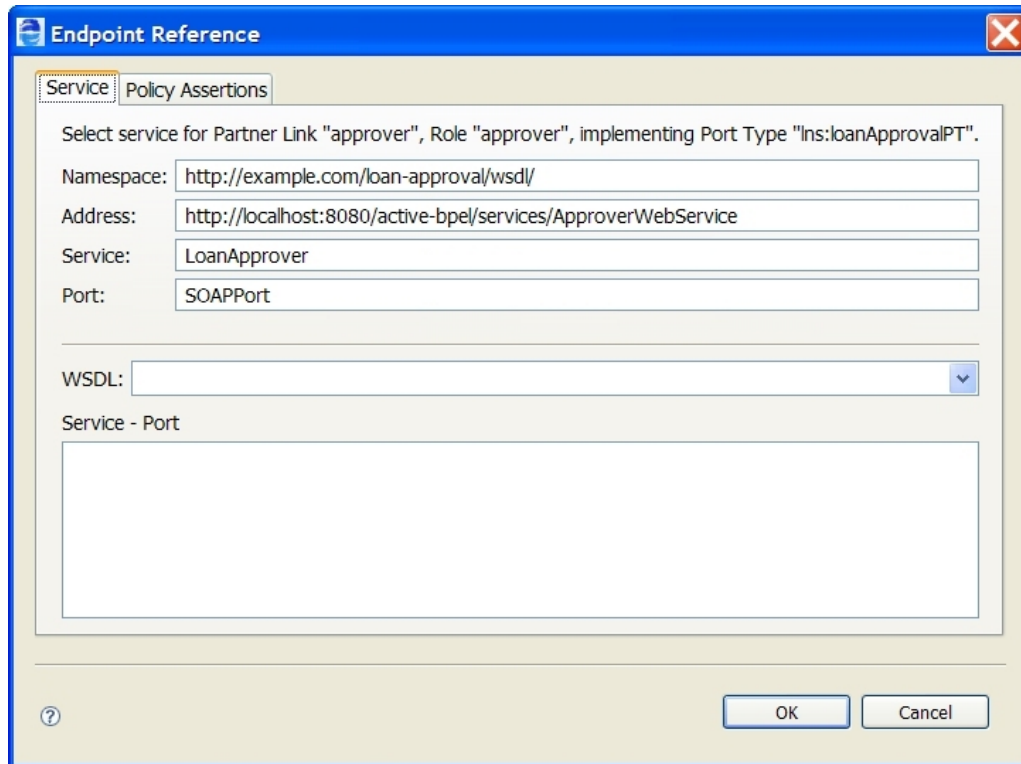
- Manually fill in the namespace, address, service, and port for the service if the binding information does not exist

- If you selected WSDL Service Port for the Invoke Handler, you cannot edit the service information. At runtime, the WSDL Service definition is always used.

See also "Using SOAP 1.1 or 1.2 Port Binding " on page 356.

See "Creating a Process Deployment Descriptor File " on page 347 for details on opening the New Deployment Descriptor dialog.

Using SOAP 1.1 or 1.2 Port Binding

As of version 7 of Process Server, support for both SOAP 1.1 and SOAP 1.2 is enabled. You can select a service containing a port binding of SOAP 1.2. If no port is specified in the service, the binding defaults to document literal and SOAP 1.1.

Note that all My Role services are deployed with both SOAP 1.1 and SOAP 1.2 bindings. Clients that want to take advantage of SOAP 1.2 can address `http://[host:port]/active-bpel/services/soap12/ [myRoleService.wsdl]`.

Adding Policy Assertions

Select a service containing the binding information for the endpoint reference. Select the Policy Assertions tab to add one or more policy assertions for the endpoint reference.

The WS-Policy specification allows Web service providers and clients to express a broad range of capabilities, requirements, and preferences in a standard way through *policy assertions*. If desired, you can add one or more policy assertions to the partner link's endpoint reference section of a process deployment descriptor file.

You can add a policy assertion to both the **Partner Role** partner link and the My Role partner link in the Deployment Descriptor wizard.

Many policy assertions specify both an *inbound* and *outbound* direction.

- An inbound assertion governs messages received by the BPEL process, such as messages for receives and replies from invoked services.
- An outbound assertion governs messages sent by the process, such as requests to an invoked service and replies matching receives.

Policy assertions apply to static endpoint references for partner roles in the PDD wizard. For a dynamic endpoint reference, you can manually add the details to the appropriate copy operation in an assign activity.

The supported policy assertions are described in the following table:

Policy	Description
WS-Security Policy Assertions	
"Authentication " on page 359	Credentials required for access to a serve
"Encryption " on page 360	Describes the parts of a SOAP message to encrypt
"Signature " on page 367	Describes the parts of a SOAP message to sign with an XML Signature, using an X.509 Certificate token
"Timestamp " on page 368	Adds a <Timestamp> element to the SOAP header of a message
Other Policy Assertions:	
"Retry " on page 365 (Partner Role)	Describes when and how many times to retry an invoked service that does not reply
"Engine-Managed Correlation " on page 361 (My Role)	A My Role policy assertion that directs the Process Server to use WS-Addressing to transmit replyTo endpoint references during transmissions to the Partner Role partner link
"WS-Reliable Messaging " on page 369	Specifies that a partner link participates in an industry-standard protocol that supports guaranteed delivery of messages
"JMS Delivery Options " on page 364 (Partner Role--*Informatica Process Developer only*)	For details, see "Using a Java Messaging Service Invoke Handler " on page 416
"HTTP Transport " on page 363	Used for REST-based invocations
"REST Enabled " on page 364 (My Role)	Used for REST-based invocations

Policy	Description
WS-Security Policy Assertions	
"SAML " on page 366	The Security Assertions Markup Language (SAML) is an OASIS standard that enables loosely coupled and federated identity integration.
"Message Validation " on page 364	Provide fine-grained validation of WSDL messages for a partner link to enable faster processing
"Web Service Timeout " on page 369	Set an amount of time to wait for a specific Web service to time out
"Invoke Recovery " on page 363 (Partner Role)	Select whether to suspend a process with a pending invoke when the process recovers from a failed server
"Send WS-Addressing Headers " on page 367 (Partner Role)	Explicitly add addressing to invokes
"WSDL Binding Reference " on page 370 (My Role)	Applicable for a my role partnerlink to refer to a WSDL binding instead of RPC or Document
"Suppress xsi type " on page 368	A workaround for suppressing schema validation in SOAP messages, useful for dealing with legacy services that cannot handle `xsi:type` attributes
"Run As User " on page 366	For subprocess invoke only. Add a user name to stand for the process initiator. Useful for a subprocess that contains a People activity that can request the name of the process initiator.

To add one or more policies to a partner link

1. In the Deployment Descriptor wizard, navigate to the Partner Links page. See "Creating a Process Deployment Descriptor File " on page 347 for details on opening the New Deployment Descriptor dialog.

2. For a partner role partner link, select static as the Endpoint Reference Type.

3. Select the Dialog button at the end of the Endpoint Reference text box.

4. Select the Policy Assertions tab, as shown here.
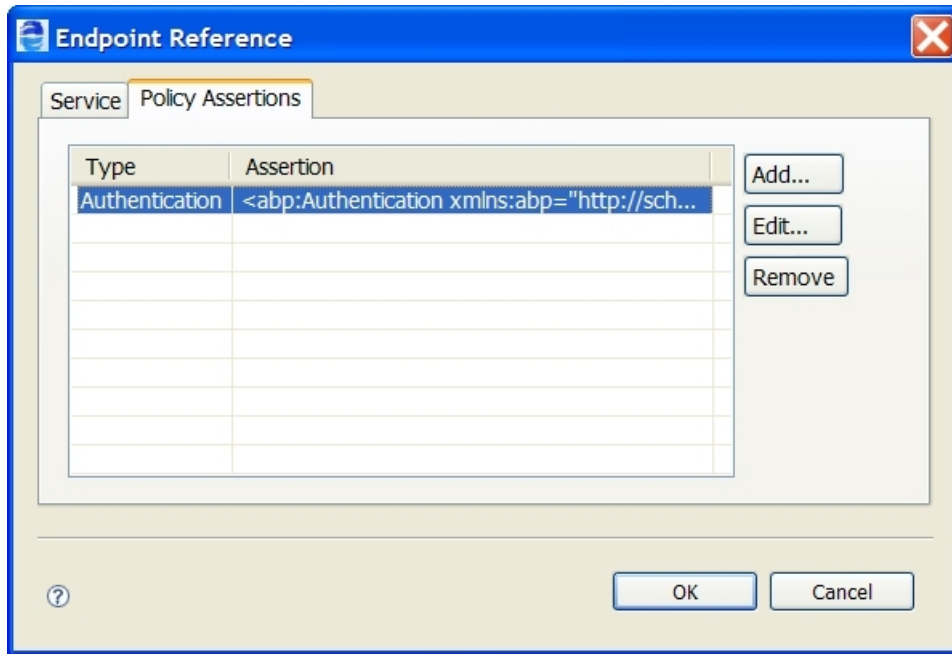
5. Select the **Add** button to open the **Policy Assertion** dialog.

6. From the Policy Template list, select the policy you wish to add, and fill in all the required information. For details, select a name from the supported policies listed at the top of this topic.

7. Select **OK**.

8. Select **Add** again to add additional policies.

9. Add My Role policy assertions by doing the following:

   a. Select a My Role partner link.

   b. Select the My Role Policy tab.

   c. Select applicable policies, as described above, for the My Role service.

Authentication

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

This policy describes the HTTP credentials required for access to a service.

Inbound authentication is for messages received for My Role services and for replies from invoked Partner Role services. Outbound authentication is for replies from My Role services and messages sent to invoked Partner Role services.

Set the following parameters:

Outbound Authentication Policy	
Username	Name of the user authorized to access the service
Password	Password for the authorized user

Preemptive HTTP Credentials	Preemptive authentication sends basic authentication credentials without being challenged by the remote host. Care must be taken to ensure that this option is only used to communicate over secure channels to prevent unnecessary disclosure of user credentials
Send `wsse:UsernameToken` in Header	If checked, credentials are sent as a OASIS WS-Security `UsernameToken` element in the SOAP Header. To include a Nonce (a nonce is an arbitrary number used only once in a cryptographic communication) with the `UsernameToken`, add a `<abp:HashPassword/>` child element to the PDD file in the PDD editor after finishing the wizard. A Nonce is used to increase the authentication security by disallowing a string to be repeated. See the example below. A `wsse:UsernameToken` header can be sent unencrypted and unsigned, provided that it is the only security policy in effect for the invoke. Otherwise, the `UsernameToken` header is signed and encrypted, which requires that you have a certificate keystore configured.
Use Cleartext Password	Select this option to store the authorized user's password in clear text in the PDD. The password is visible and readable in the PDD file.
Inbound Authentication Policy	
Require `UsernameToken` when receiving messages	Select this to require authentication on messages received by a My Role or Partner Role service

Example:

```
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing"
                       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                       xmlns:ns5="http://www.example.org/Hello/"
                       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <wsa:Address>http://localhost:8081/active-bpel/services/Hello</wsa:Address>
    <wsa:Metadata>
        <wsa:ServiceName PortName="HelloSOAP">ns5:Hello</wsa:ServiceName>
        <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                    xmlns:abp="http://schemas.active-endpoints.com/ws/2005/12/policy">
            <abp:Authentication direction="out">
                <abp:User>aeadmin</abp:User>
                <abp:Password>yH6CJei+D+s=</abp:Password>
                <abp:HashPassword/>
            </abp:Authentication>
        </wsp:Policy>
    </wsa:Metadata>
</wsa:EndpointReference>
```

Encryption

Select this policy assertion as described in ["Adding Policy Assertions " on page 357](#).

An encryption policy describes the parts of a SOAP message to encrypt, in compliance with the processing rules of the XML Encryption specification [XMLENC].

Each specified original element or element content in the message is removed and replaced by the resulting encrypted element.

- **Inbound** encryption is for messages received for My Role services and for replies from invoked Partner Role services. It indicates that the My Role partner role accepts encryption and will de-encrypt messages received.

- **Outbound** encryption is for replies from My Role services and messages sent to invoked Partner Role services.

Set the following parameters:

Encryption Parts Attributes	
alias	Optional keystore alias used to retrieve the key for encryption. The default is the alias specified in the crypto properties file.
Content Name	Message part or element to be encrypted
Content Namespace	Target namespace of the message part or element

Example

```
<abp:EncryptionParts alias="keystore_alias">
  <abp:Element
   namespace="http://docs.oasis-open.org/wss/2004/01/
     oasis-200401-wss-wssecurity-secext-1.0.xsd"
   name="UsernameToken"/>
</abp:EncryptionParts>
```

As a message consumer, Process Server service endpoints accept and consume messages that conform to options deemed allowable under WS-I guidelines. As a message producer, Process Server supports only the recommended algorithms, references and identifiers.

The following algorithms are used within the data encryption of SOAP messages. Additional algorithms will likely be added and supported in future releases, based on WS-I recommendations and customer demand.

The supported token types are as follows:

- **X.509 Token**
  Direct Binary Reference (send and receive): Preferred method, used where possible.

  Issuer Serial (send and receive): Preferred external reference method if direct not possible.

  X509 Identifier (receive only)

  Subject Key Identifier (receive only)

  Embedded Token References (receive only)

- **Symmetric Data Encryption Algorithms:**
  http://www.w3.org/2001/04/xmlenc#tripledes-cbc (send and receive)

  http://www.w3.org/2001/04/xmlenc#aes128-cbc (receive only)

  http://www.w3.org/2001/04/xmlenc#aes256-cbc (receive only)

- **Asymmetric Key Transport Algorithms:**
  http://www.w3.org/2001/04/xmlenc#rsa-1_5 (send and receive)

  http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (receive only)

- **Signature Digest Algorithm:**
  http://www.w3.org/2000/09/xmldsig#sha1 (send and receive)

- **Signature Algorithm:**
  http://www.w3.org/2000/09/xmldsig#rsa-sha1 (send and receive)

- **Cannonical XML Transform Algorithm:**
  http://www.w3.org/2001/10/xml-exc-c14n# (send and receive)

Engine-Managed Correlation

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

There are two ways that Process Server can correlate messages that are received by a BPEL process: correlation sets and engine-managed correlation policy assertions. A correlation set uses message properties defined within a WSDL and BPEL process. Engine-managed correlation relies on WS-Addressing references in the SOAP header of a message to correlate a received message with the correct process instance.

If you use engine-manage correlation, you need only specify a policy assertion on a My Role partner link. Select a My Role partner link that is associated with a Partner Role in the same partner link. The engine-managed correlation policy assertion directs the Process Server to use WS-Addressing to transmit replyTo endpoint references during transmissions to the partner.

Policy Assertion Example

```
<myRole allowedRoles="" binding="RPC"
    service="ManagedCorrelationServiceA">
  <wsa:Metadata>
   <wsp:Policy xmlns:abp="http://schemas.
      active-endpoints.com/ws/2005/12/policy"
        xmlns:wsp="http://schemas.xmlsoap.org
        /ws/2004/09/policy">
    <abp:engineManagedCorrelationPolicy/>
   </wsp:Policy>
  </wsa:Metadata>
</myRole>
```

How Partner Services Handle Engine Managed Correlation Messages

Often a partner service implemented the WSA SOAP Binding details to automatically handle engine managed correlation messages. The service extracts information from the SOAP header and uses it to reply back to Process Server.

The message Process Server sends to the service is constructed similarly to the following example. Note that the SOAP header contains a `<wsa:ReplyTo>` element including a `<wsa:ReferenceProperties>` element containing the conversation Id for the correlation:

Outbound Message from Process Server

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
      /soap/envelope/" xmlns:xsd="http://www.w3.org/2001
      /XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    ...
   <wsa:ReplyTo xmlns:abx="http://www.activebpel.org/bpel/extension"
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
    <wsa:Address>http://localhost:8080/active-bpel/services
        /ManagedCorrelationServiceA</wsa:Address>
          <wsa:ReferenceProperties>
        <abx:conversationId>1:/process/partnerLinks
           /partnerLink[@name='requestPLT']
              </abx:conversationId>
          </wsa:ReferenceProperties>
   </wsa:ReplyTo>
    ...
  </soapenv:Header>
  <soapenv:Body>
    ...
  </soapenv:Body>
</soapenv:Envelope>
```

When a service receives a message, it should extract the SOAP header `<wsa:ReferenceProperties>` and reply with a similarly formatted SOAP header. If the service you are calling is not replying as expected, use the following example to understand the expectations of Process Server for receiving an engine managed correlation message.

Inbound Message Process Server Expects

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org
     /soap/envelope/" xmlns:xsd="http://www.w3.org/2001
     /XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Header>
    ...
    <abx:conversationID
         xmlns:abx="http://www.activebpel.org/bpel/extension">
       1:/process/partnerLinks/partnerLink[@name='requestPLT']
    </abx:conversationId>
   ...
 </soapenv:Header>
 <soapenv:Body>
    ...
 </soapenv:Body>
</soapenv:Envelope>
```

For details on correlation sets, see Correlation.

HTTP Transport

HTTP Transport

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

You can add this policy to a Partner Role. It is not applicable for a My Role. Depending on the REST service requests from the process, you may wish to set one or more of the HTTP transport sender timeout values.

Note: This policy currently works only in conjunction with REST-enabled processes.

Client Connection Timeout	The timeout until a connection is established. A value of zero means the timeout is not used. Zero is the default.
Connection Pool Manager Timeout	Set the connection acquisition timeout value. The default is not defined.
Socket Timeout	The default socket timeout (SO_TIMEOUT) in milliseconds that is the timeout for waiting for data. A timeout value of zero is interpreted as an infinite timeout. This value is used when no socket timeout is set in the HTTP method parameters.
TCP no delay	Enable, if used. The TCP no delay parameter controls TCP (Transmission Control Protocol) packet batching. The default value is 1, which means that TCP packets are not batched. Determines whether Nagle's algorithm is to be used. (The Nagle's algorithm tries to conserve bandwidth by minimizing the number of segments that are sent.) To decrease network latency and increase performance, Process Server disables Nagle's algorithm (by enabling TCP_NODELAY). Data is sent earlier, at the cost of an increase in bandwidth consumption and number of packets.
Redirect with GET	Instruct the browser to load a different page using an HTTP GET request.
XML MIME Types	To interpret data correctly, add additional MIME types, such as `image/svg+xml`.

Invoke Recovery

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

When a server goes down and then comes back online, Process Server recovers and resumes active processes automatically. However, for a pending invoke activity, you may want to suspend the process to

inspect whether the service partially or fully completed its operation. In the Process Console, you may wish to manually restart the suspended process using the input message data.

If a server terminates unexpectedly, processes may be executing. Because they may not have saved process state to the database, the server must execute parts of a process to bring it back to the state it was in. Journal entries record the steps between process state being saved, and they are re-played during recovery. When a failure occurs, another node in the cluster may attempt to recover the process after a designated grace period.

You can add a policy assertion to the endpoint reference for the partner role of the partner link associated with an invoke activity to specify whether to suspend the process if that invoke activity is in a pending (executing) state during process recovery.

Select a setting for suspend process:

- **Default**. If this optional attribute is not present, then invoke recovery handling is specified by the Suspend process on invoke recovery setting on the Engine Properties tab of the Engine Configuration page.
- **True**. Suspend a process when a pending invoke is found during process recovery
- **False**. Do not suspend a process when a pending invoke is found during process recovery.

JMS Delivery Options

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

For details on JMS Messaging, delivery options, and examples of adding attributes to delivery options, see "Using a Java Messaging Service Invoke Handler " on page 416.

Message Validation

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

By default, the Process Server engine validates the data loaded into process variables against the WSDL schema prior to process execution. This validation property is a global setting that can be disabled/enabled in the Process Console.

If desired, you can add a policy assertion to an endpoint reference for fine-grain control of message validation. The policy assertion overrides the global setting in the engine.

By disabling message validation, you are enabling faster processing. However, be aware that there are many cases where invalid data can be introduced into the process that will not be caught unless you provide fault handling or validate activities. With message validation enabled, the process will throw a fault if invalid data is detected.

Select one of the following:

- **None**. Message validation is disabled.
- **Both**. Message validation is enabled for input and output messages.
- **In**. Message validation is enabled for input messages.
- **Out**. Message validation is enabled for output messages.

REST Enabled

Select this policy assertion as described in "Adding Policy Assertions " on page 357. You can set this policy on a My Role partner link to provide a REST binding.

This policy provides a context path for the partner link. The context value of the REST-enabled policy is the value used to lookup the process and partner link names registered with the deployment manager.

On deployment, a My Role partner link containing a REST Enabled policy registers the mapping between the REST path and the process and partner link names.

The *Usage* and *Description* fields are optional. You can add the BPEL process name in the Description field and add REST service parameters in the Usage field.

If you fill in these fields for your REST-enabled processes, you can then retrieve a list of deployed REST services, listed by BPEL process and the details you added in the Usage field.

For example, at runtime, the following request URL returns a list of services that have included a description and optionally usage parameters:

```
http://localhost:8080/active-bpel/services/REST?
```

Retry

Select this policy assertion as described in *Adding Policy Assertions*.

This policy describes when and how many times to retry an invoked service that does not reply.

During process execution, an invoked service can not reply, and, assuming that the service responds with a fault, you can specify how you want to handle the faulting execution.

You can specify the relatively simple policy that the server automatically retries the invocation a certain number of times.

Alternately, you can create a BPEL process to handle retries. For details, see *Retry-Policy Service*.

Add the retry attributes as follows:

Assertion parameter	Description	Default
Interval	Amount of time in seconds to wait between retries	None. You must add a value. Ignored if a service name is specified.
Attempts	Number of times to retry the service	None. You must add a value. Ignored if a service name is specified.
Service	Name of a service implemented by the process. The service is a BPEL process based on a required WSDL operation.	If you specify a service, the attempts and interval values are ignored.
faultList	Faults to include for retries	All faults are retried
faultExclusionList	Faults to exclude for retries	
On Failure	The state for an invoke activity when the retries are exhausted: Faulted (the default) or Suspended.	Fault

XML Syntax

```
<abp:retryPolicy attempts="xsd:int"?
    interval="xsd:int"?
    service="NCName"?
    faultList="QNameList"?
    faultExclusionList="QNameList"?
    onFailure="fault"/>
```

Example 1: Policy with a service and fault list:

```
<wsa:EndpointReference
    xmlns:s="http://www.activebpel.org/services/retrytests">
  <wsa:ServiceName PortName="retrytesterServicePort">
```

```
       s:retryTesterService</wsa:ServiceName>
     <wsp:Policy
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
      xmlns:abp="http://schemas.active-endpoints.com/ws
         /2005/12/policy">
      <abp:retry service="retryCheckerService">
        faultList="{http://xml.apache.org/axis/}*
           {http://www.active-endpoints.com/2004/06/bpel
              /extensions/}*
           {http://www.activebpel.org/services
              /retrytests}*"/>
     </wsp:Policy>
   </wsa:EndpointReference>
```

Example 2: Policy with attempts and fault exclusion list:

```
<wsa:EndpointReference
     xmlns:s="http://www.activebpel.org/services/retrytests">
   <wsa:ServiceName PortName="retrytesterServicePort">
     s:retryTesterService</wsa:ServiceName>
   <wsa:Metadata>
    <wsp:Policy
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
     xmlns:abp="http://schemas.active-endpoints.com/ws/
      2005/12/policy">
    <abp:retry interval="1" attempts="3">
       faultExclusionList="{http://xml.apache.org/axis/}*"/>
    </wsp:Policy>
   </wsa:Metadata>
</wsa:EndpointReference>
```

Run As User

Select this policy assertion as described in *Adding Policy Assertions*.

This policy is applicable only to the process and subprocess invoke handlers, described in *Partner Role Invoke Handlers*. Use this policy assertion to override the default value of process initiator.

By default, a process instance is associated with a user role called *process initiator*. Process Server populates the identity of process initiator with the Process Server credentials of the calling process, if the server is secured, or with *anonymous*, if the server is not secured.

The Run As user value is a string. It does not need to be an actual user from your Identity service.

SAML

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

The Security Assertions Markup Language (SAML) is an OASIS standard that enables loosely coupled and federated identity integration. SAML standardizes how identity-related security information can be communicated between policy domains.

SAML assertions are usually transferred from identity providers (the my role partner link) to service providers (the partner role partner link). Assertions contain statements that service providers use to make access control decisions.

SAML 1.1 and 2.0 are the currently supported versions. For details about SAML, refer to OASIS Security Services (SAML) TC at www.oasis-open.org.

Direction	- **Out**. Typically selected for Partner Role partner links. Messages sent to partner service will be trusted messages - **In**. Typically selected for My Role partner links. Messages sent back to the process from partner service are accepted as trusted messages - **Both** can be needed when the transport mechanism is other than SOAP over HTTP, such as SOAP over JMS. Send and receive trusted messages.
Version	SAML Version to use
Subject Name	(Optional) For outgoing messages, add a subject to indicate the the user associated with the identity information. For example, you can enter the distinguished name from your LDAP service.
Confirmation Method	For outgoing messages, select a method: - **sender-vouches**: If trust is already established with a SSL certificate, then a digital signature is not required, and you can use sender-vouches. - **holder of key**: If trust has not been established, you can select holder of key to indicate that the proof of trust is sent through digital signatures within the assertion itself.
Authentication Method	For outgoing messages, select a method used to authenticate the subject (to determine if the information in the assertion refers to the party making the current request). The default is `urn:oasis:names:tc:SAML:1.0:am:unspecified`. For details about using the other options, refer to the SAML Specification at the address given in the introduction of this topic.

Send WS-Addressing Headers

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

You can add this policy assertion to a partner role partnerlink. The policy is an extension element to allow endpoints to explicitly declare if addressing should be used when making requests. Addressing is automatically added to invokes if you add this policy.

For more information, see `http://www.w3.org/TR/ws-addr-metadata/#indicatinguse`.

Signature

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

A signature policy describes the parts of a SOAP message to sign with an XML Signature, using an X.509 Certificate token to allow for verification and trust of the signed information.

- **Inbound** signature is for messages received for My Role services and for replies from invoked Partner Role services. It indicates that the My Role partner role accepts signed message content and will verify the signature.

- **Outbound** signature is for replies from My Role services and messages sent to invoked Partner Role services.

Set the following parameters:

SignatureParts alias	Optional keystore alias used to retrieve the key to sign. The default is the alias specified in the crypto properties file.
Content Name	Message part or element to be signed
Content Namespace:	Target namespace of the message part or element

Example

```
<abp:SignatureParts alias="keystore_alias">
   <abp:Element
    namespace="http://docs.oasis-open.org/wss
       /2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    name="UsernameToken"/>
</abp:SignatureParts>
```

As a message consumer, Process Server service endpoints accept and consume messages that conform to options deemed allowable under WS-I guidelines. As a message producer, Process Server supports only the recommended algorithms, references and identifiers.

The following algorithms are used for signing SOAP message parts.

- **X.509 Token**
  Direct Binary Reference (send and receive): Preferred method, used where possible.

  Issuer Serial (send and receive): Preferred external reference method, if direct not possible.

  X509 Identifier (receive only)

  Subject Key Identifier (receive only)

  Embedded Token References (receive only)

- **Signature Digest Algorithm:**
  http://www.w3.org/2000/09/xmldsig#sha1 (send and receive)

- **Signature Algorithm:**
  http://www.w3.org/2000/09/xmldsig#rsa-sha1 (send and receive)

- **Cannonical XML Transform Algorithm:**
  http://www.w3.org/2001/10/xml-exc-c14n# (send and receive)

Suppress xsi type

Select this policy assertion as described in .

You can add this policy assertion to either a partner role partner link or a my role partner link to suppress `xsi:type` attributes in SOAP messages.

Enabling this policy assertion effectively disables schema validation on messages for an invoke or receive since they cannot be validated without the type attributes.

This policy assertion should only be used as a workaround for legacy services that cannot deal with `xsi:type` attributes and cannot be fixed or upgraded.

Timestamp

Select this policy assertion as described in .

This assertion adds a `<Timestamp>` element to the SOAP header of a message as defined by the OASIS WS-Security 1.0 specification. Usually, a timestamp is signed, as described in the ["Signature " on page 367](#) policy assertion.

The Time To Live attribute indicates the limit on receiver side that the timestamp is validated against.

The Direction attribute indicates whether the timestamp is validated only at the invoked service, only on the server, or both. The values can be in (server), out (invoked service), or both.

Web Service Timeout

Select this policy assertion as described in ["Adding Policy Assertions " on page 357](#).

For performance reasons, a reply activity matching a receive, as well as synchronous invokes, are timed out if they do not execute on Process Server within 10 minutes. If you are receiving timeout errors, you can specify a greater amount of time to wait before a process is timed out due to a reply or synchronous invoke activity not executing within 10 minutes. The default is 600 seconds.

Enter a time out value in seconds. A value of zero indicated unlimited wait time.

WS-Reliable Messaging

Select this policy assertion as described in ["Adding Policy Assertions " on page 357](#).

The WS-Reliable Messaging policy assertion specifies that a partner link participates in an industry-standard protocol that supports guaranteed delivery of messages.

The OASIS business standards consortium's WS-Reliable Messaging standard provides a protocol for the guaranteed delivery of messages. Process Server supports this standard, allowing BPEL processes to use the protocol to identify, track, and manage the reliable delivery of messages.

Process Server implementation of reliable messaging is based on the following description.

After sending a message, Process Server waits for the amount of time set in the Acknowledgement Interval to get an acknowledgement message. If Process Server times out and has not received an acknowledgement, the Process Server waits for the amount of time set in the Base Retransmission Interval before resending. If the sequence has not completed within the amount of time set in the Inactivity Timeout, Process Server considers the sequence terminated.

Add the WS-Reliable Messaging attributes as follows:

Assertion parameter	Description	Default
Acknowledgement Address	Address to be used for acknowledgements	`http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous` This address represents the anonymous role as defined by WS-Addressing and indicates that acknowledgements are sent synchronously.
My Role binding	SOAP binding style that this partner link uses. Specify the binding shown on the Partner Links page of the PDD wizard.	Document Literal

Assertion parameter	Description	Default
Inactivity timeout	The amount of time the Process Server waits before a sequence is considered terminated. This can be set to a relatively long period of time.	240000 (In milliseconds)
Base transmission interval	The amount of time Process Server waits before resending an unacknowledged request.	20000 (In milliseconds)
Acknowledgement interval	The amount of time Process Server waits before a message is considered unacknowledged. This is usually a relatively short value.	10000 (In milliseconds)

WSDL Binding Reference

Select this policy assertion as described in "Adding Policy Assertions " on page 357.

You can select the binding style of Policy-Driven for a my role partner link and then select this option to refer to a WSDL binding for service binding details.

Type in a Namespace URI and the associated binding details. Or select a WSDL located in the Workspace with matching binding details.

Adding Indexed Properties

Add indexed properties for use in quickly finding certain processes running on the server.

An indexed property is a process variable property that is populated with the variable (or variable part) value during process execution. If you add indexed properties to the PDD, on the Process Console, you can select indexed properties as a search filter for process instances.

Also you can use indexed properties to define a business process event. For details, see Chapter 32, "Business Event Processing " on page 433.

For example, you can find all faulting processes where the customerId is 101. In this example, the indexed property name is `customerId` and is based on a process variable of type string. "101" is the data value at the time the process is suspended.

Indexed properties allow for fast searching of important data items at runtime.

Add indexed property details as follows:

Name	Indexed property name. This name appears in the Indexed Property list in the Process Console.
Type	Select the property type from: boolean, date, double, string. Match the type to the variable, part or query value
Variable Path	Full process variable path. Select from the picklist, or select the Dialog button to open the Find a BPEL Variable dialog. If you have a long list of variables, the dialog is more convenient.
Part	Process variable part for message type variables
Query	Process variable part detail (optional)

Eventing tab of the PDD

For details, see [Business Event Processing](#).

Viewing References

The References tab of the PDD Editor displays the resources in use in the deployment descriptor. When you deploy the PDD, the BPEL process and all resources listed on this tab are deployed to the server.

See also ["Creating a Process Deployment Descriptor File " on page 347](#).

People tab of the PDD

For details, see the Human Task sections in this help.

Using the PDD Editor Source View

The Process Deployment Descriptor editor validates the `.pdd` file against a schema. If the editor detects errors, it generates errors in the Error Log when you save the file.

You may wish to edit the file in the Source tab, which is a text editor, for convenience. For example, you can specify actual endpoint locations for each Web service, or you type in a URN.

To add static endpoint references:

1. For each partner link, do the following:

   - Replace `FILL_IN_NAMESPACE` with the target namespace from the WSDL file where the partner link type is defined.

   - Replace `FILL_IN_ADDRESS_URI` with the URI of the service as specified in the `<service>` section of the WSDL file for the service. Alternately, you can specify a URN, as described in ["Specifying a Replaceable URN URL for an Endpoint Reference " on page 345](#).

   - Replace `FILL_IN_PORT_NAME` with the service port name as specified in the `<service>` section of the WSDL file for the service.

   - Replace `FILL_IN_SERVICE_NAME` with the service name as specified in the `<service>` section of the WSDL file for the service.

2. Save the file.

Tips for using the PDD Editor Source tab:

- Be sure to check the Error Log for a list of errors and warnings.

- You can display line numbers in the editor, if desired. Right-mouse click in the Source view to select **Preferences**. Enable Show line numbers and other Text Editor preferences as desired.

See also ["Creating a Process Deployment Descriptor File " on page 347](#).

Creating and Deploying a Business Process Archive Contribution

View the selected deployment descriptor file(s) and other project resources to deploy. On the Catalog Resources page, view the list of selected and excluded resources to ensure your contribution contains all files to be managed as a unit. Upon export, deploy the file and save a script to redeploy the .bpr file.

To deploy your process to A Process Server, you package a contribution of processes and resources in a business process archive file (.bpr file), which is similar to a Web archive file.

Before creating a .bpr file, you must create a .pdd file, described in *Creating a Process Deployment Descriptor File*. Optionally, you can additional resources, such as Process Central forms and tasks.

We recommend that you create only one BPR file per project. The processes and resources are managed as a unit of files on the server. For details, see *Managing Deployment Contributions*.

To create a .bpr file:

1. If you have created and used a project reference for the current project, be sure to deploy that contribution first. Project dependencies must be deployed to the server first.

2. Select **File > Export > Orchestration > Contribution-Business Process Archive**.

3. Notice that all process deployment descriptor files (.pdd files) are pre-selected to include in the archive, as shown in the following example.
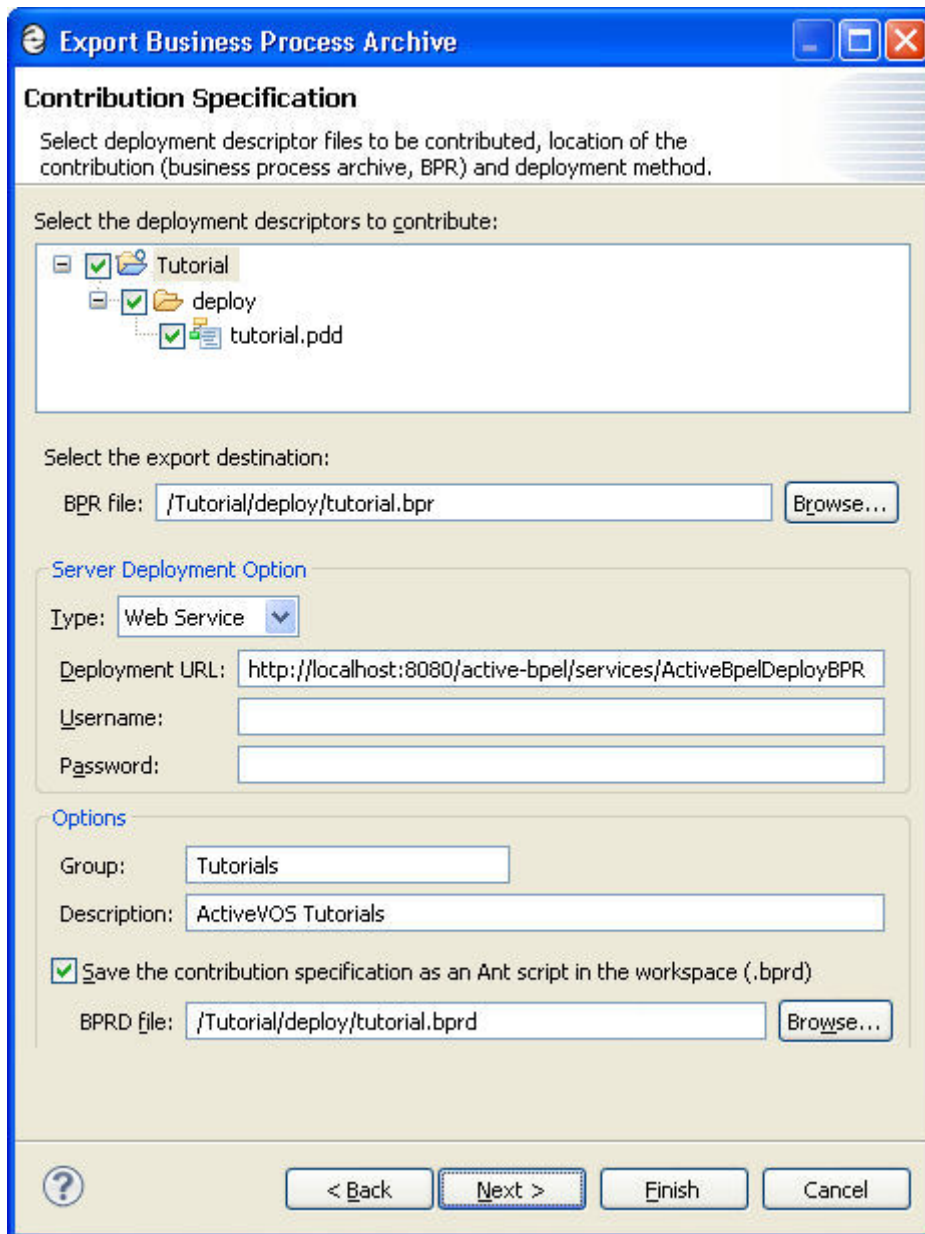
4. Select the export location and filename for the BPR file.

5. Select a **Server Deployment Type**:

   - **None**. Select this option if you do not wish to deploy your BPR archive upon completion of the export.

   - **File**. Select this option to save the archive upon export to a file system location. In the **Deployment location** field, browse to any folder.

   - **Web Service**. Select this option if your server is running and you want to automatically deploy to it. Is your server is secured, provide the username and password for access to it.

6. If desired, type in a **Group** name and description for this contribution. On the Process Console you can use the group name as a selection filter for listing files. The description appears on the Contribution detail page of the Console.

7.   If desired, select the option to **Save the contribution specification as an Ant script in the workspace**, and select a location for this file, such as the Orchestration Folder's deploy folder. For details on this option, see *Using a BPRD Script to Regenerate and Deploy a BPR Contribution*.

8.   Click **Next** to view a list of **catalog resources**. See the following topics.

   • Contribution Preference

   • Deploying Project Dependencies and Viewing Excluded Dependencies

   • Deploying Additional Resources

9.   Click **Finish** to create and export the .bpr file and optional .bprd file.

10.   Refresh the project to view the new BPR file.

See also Managing Deployment Contributions.

Deploying Project Dependencies and Viewing Excluded Dependencies

As described in *Creating and Deploying a Business Process Archive Contribution*, you begin a deployment at the project level, by including the process deployment descriptors (PDDs) of the processes in your project. The Export Business Process Archive Wizard then shows you all project resources pre-selected and project references excluded from deployment, as shown in the example.
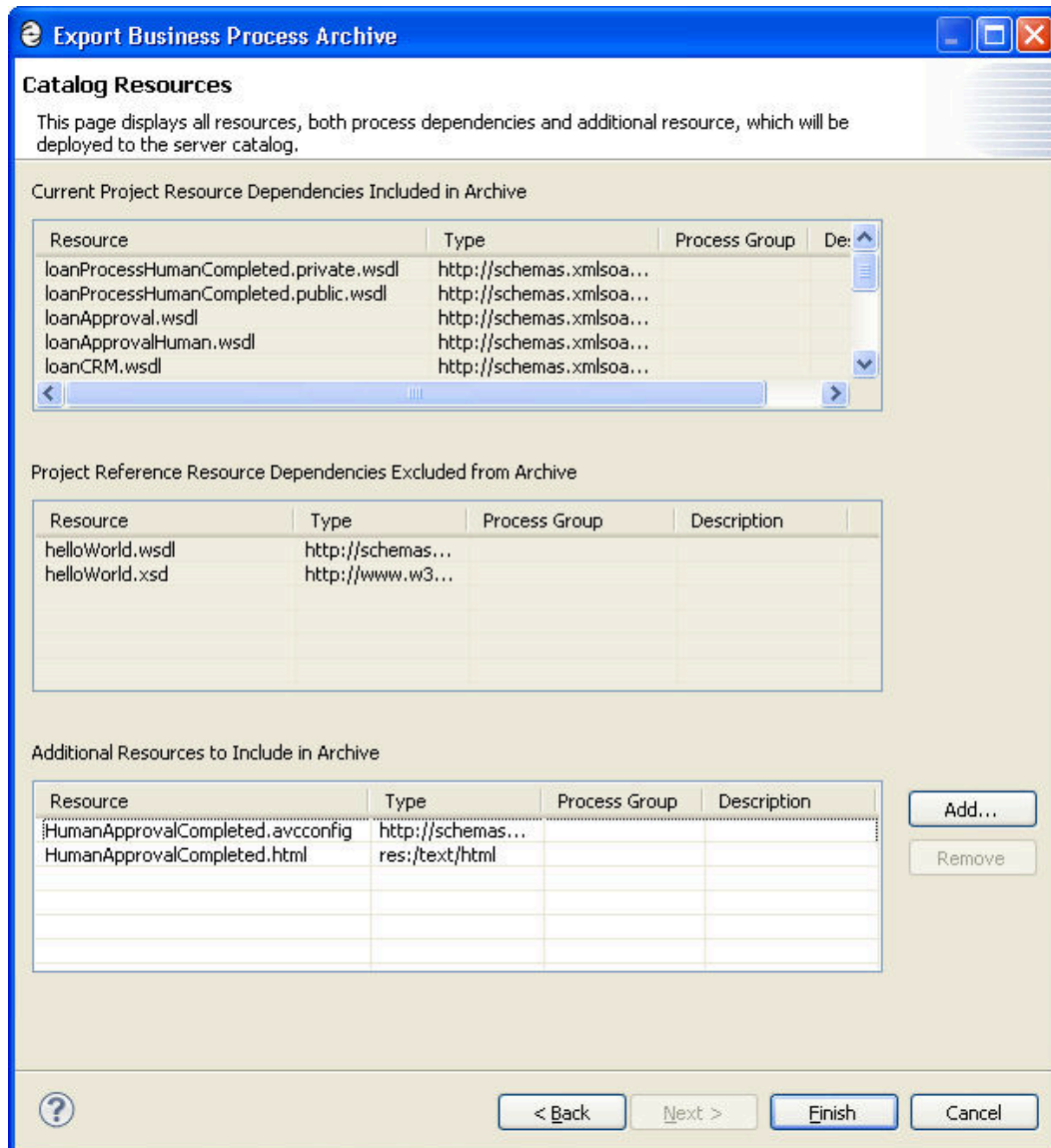
The parts of this dialog are described as follows:

Project Resource Dependencies

The current project resource dependencies are deployed. They include the following types:

- WSDL and schema from this project
- WSDL and schema imported from another workspace project if you do not have project references specified
- Human task HTML forms
- (not displayed) XSL files used in the `doXSLTransform` function, based on the location of the file you specified as a parameter to the function
- Java jars specified in the PDD

Project Reference Dependencies Excluded from Archive (not deployed)

You must deploy these resources in a contribution prior to deploying this contribution.

Project References specified in the project properties and used within the process are listed on the Catalog Resources page. For example, if you create a participant based on a referenced WSDL, you will see the WSDL listed in the excluded list. For details on the advantage of this approach, see *Using Project References*.

Additional Resources

Additional resources are automatically pre-selected, based on the *Contribution Preference*.

You can deploy more resources as described in *Deploying Additional Resources*.

Deploying Additional Resources

Add resources to deploy to the server. View the selected deployment descriptor files and other project resources to deploy. On the Catalog Resources page, view the list of selected and excluded resources to ensure your contribution contains all files to be managed as a unit. Upon export, deploy the file and save a script to redeploy the .bpr file.

For an overview of best practices for deployment, see "Deploying Project Dependencies and Viewing Excluded Dependencies " on page 374.

As you develop a process, there are two kinds of resources you can add: dependent and non-dependent resources. Dependent resources include WSDL, schema, human task forms, XSL for doXSLTransform, and Java jars specified in the PDD. Dependent resources can be deployed in the same BPR contribution containing the process PDD or in a referenced project.

You may want to deploy XQuery modules in a separate BPR.

The resources you may want to deploy in a separate BPR include:

- Reports. For details, see Creating Reports for Process Server and Central.
- Central Configuration. Customize the Process Central client application. For details, see What is Process Central?.
- Binary and HTML files. For process request forms, you can export HTML, language files, images, and style sheets. For details, see "What is Process Central " on page 441.
- XQuery Modules. Refer to "Writing XQuery Functions " on page 404.

To deploy additional resources to the server:

1. Ensure that the server is running, and ensure that your resource file is in a Workspace project.
2. Select **File > Export > Orchestration > Contribution- Business Process Archive File**.
3. If the resources are not directly imported into the process, you can skip the process deployment descriptor files (.pdd files) to include in the .bpr archive.
4. Select the export location and filename for the .bpr archive.
5. In the Deployment section, select Web Service as the Type.
6. On the Catalog Resources page, if needed, select **Add**.
7. Select a Resource Type and Location. Select a resource from the Project Explorer.
   For Other, add a URI. The URI can be any reference desired, preferably related to the resource type. For example, an XML resource URI should be the root namespace of the XML document.

   See also "Deploying Forms Reports and Configuration Files" on page 460.
8. In the Resources list, do the following:
   - Optionally, type in a new or existing Group Name. The Server can display resources by groups.
   - Optionally, type in a Description. The description is displayed in the Catalog Resources list on the server.
9. Add additional resources as desired.

Managing Deployment Contributions

Add resources to deploy to the server.

A *contribution* is a deployed business process archive (BPR), managed on the server as a unit of files. Rather than deploy and replace individual BPEL processes and resources, you deploy both current and updated files as a unit that is easily managed on the server.

If you are deploying one of the Process Developer sample projects, it should only have one sample-based contribution.

Contribution Name

Each contribution has a project-location-based identifier. On the Contributions page of the Process Console, you see a list of `project:/myProject` files. Each contribution is named according to your workspace project name. In the `.BPRD` file, you can view an attribute named `contributionURI` that is based on the project name: `project:/myProject`. This location hint is a helpful way to track deployments on the server.

Contribution States

Contributions have the following states: online, offline, and offline pending.

Your first deployment is version 1.0 and is *online*. As you update and redeploy your files, older contributions go *offline pending* and the new one comes online.

Only one version is online. The older versions become *offline* when all running processes associated with the *offline pending* version have completed.

Server Management of Contributions

Using contributions, it is easy to do the following tasks on the server:

- Delete all old process instances and old resources by deleting the contribution.
- Maintain your own resources that do not collide with those of other developers.
- Rollback the current (online) contribution to an earlier version and vice versa.
- Easily access the deployment log for the contribution on the Contribution detail page.

Contribution Deployment Tips

To maintain the ease of use for contributions, be sure to follow these deployment tips:

- Always deploy a project reference contribution prior to deploying the parent contribution. For details, see *Using Project References*.
- Maintain one BPR per project. Include all deployable files in the BPR. Use the Export Wizard to redeploy.
- As you update project resources, re-deploy the entire project
- As you add a new resource, re-deploy the entire project.
- As you delete resources, re-deploy the entire project.

Using a BPRD Script to Regenerate and Deploy a BPR Contribution

Select a workspace project to store the Ant script that you can execute for future re-deployments of the `.bpr` archive.

When you create a `.bpr` file, as described in ["Creating and Deploying a Business Process Archive Contribution " on page 372](#), you have the option to create an Ant script (a .bprd file) that you can use to quickly regenerate and deploy a .bpr file.

Use the BPRD Ant script to:

- Update a `.bpr` file after modifying a BPEL or process deployment descriptor file
- Add deployment path information for automatically deploying a .bpr file
- Add appropriate entries for resources that are not in a workspace project, such as an XSL style sheet

You can execute the BPRD Ant script from within Process Developer or on the command line outside of Process Developer.

For details, see:

Running a BPRD Ant Script from within Process Developer

When you create a `.bpr` file, as described in ["Creating and Deploying a Business Process Archive Contribution " on page 372](#), you have the option to create an Ant script (a `.bprd` file) that you can use to quickly regenerate and deploy a `.bpr` file.

Automatically Deploying a .bpr File

If you did not select a Deployment Type when you created the `.bpr` archive, you can add deployment information to the BPRD script and execute the script.

1. From the Project Explorer, double-click the `.bprd` file to open it.
2. Follow the comments in the script to provide File or Web Service deployment information. You must specify a valid archive deployment location and enable the deploy target.
   For example, for the archive.deployment value, type in:

   ```
   http://localhost:8080/active-bpel/services/ActiveBpelDeployBPR
   ```

3. Save the file.
4. Ensure that the server is running, and then in the Project Explorer, select the right-mouse menu for the `.bprd` file and select **Execute**.
   Process Manger informs you when the new `.bpr` is deployed.

Regenerating a .bpr File

You must update a `.bpr` file after you have modified one or more files in the original archive. If your archive contains many BPEL and `.pdd` files, you may find it quicker and more convenient to update the archive by using the Ant script you created when you first created the `.bpr` file.

If you want to add any new processes and their resources to the archive, you must create a new `.bpr` using the **File > Export** option or modify the .bprd file by hand to add syntactically correct references to the new files. To modify the `.bprd` file by hand, you must be familiar with the Ant scripting language.

To regenerate a .bpr file:

1. In the Project Explorer, select the `.bprd` file you created the first time you created the business process archive (`.bpr`) file.

2. Right-mouse click and select **Execute**.
   Process Developer informs you when the new `.bpr` is generated.

Running a BPRD Ant Script from the Command Line

When you create a `.bpr` file, as described in <u>"Creating and Deploying a Business Process Archive Contribution " on page 372</u>, you have the option to create an Ant script (a `.bprd` file). You can execute this file within Process Developer or on the command line.

To execute the BPRD Ant script from the command line, you must set up your Ant environment and install the Ant Runtime component of Process Developer.

There are sometimes network issues if your are using JAVA 7 and IpV6. If you experience issues with BPR deployment or cluster communications, you should use the `-Djava.net.preferIPv4Stack=true` flag when starting your server.

To set up Ant, you must make files available to your Ant environment. Choose one of the following methods.

Method One: Set up your Ant environment by copying jars to your Ant installation

Ensure that the following files are in your `{$ant.home}\lib` directory

- `axis.jar`
- `bpr_tasks.jar`
- `jaxrpc.jar`
- `commons-logging.jar`
- `commons-discovery.jar`
- `saaj.jar`
- `wsdl4j.jar`

Method Two: Install the B-unit Runtime and edit your BPRD file to make the changes described below

Change the following lines:

to this:

Deployment Complete

This dialog provides a quick look at the deployment results for your process files.

The **Deployment Complete** dialog displays the results of deploying a .bpr file to A Process Server.

The following illustration shows an example of this dialog.
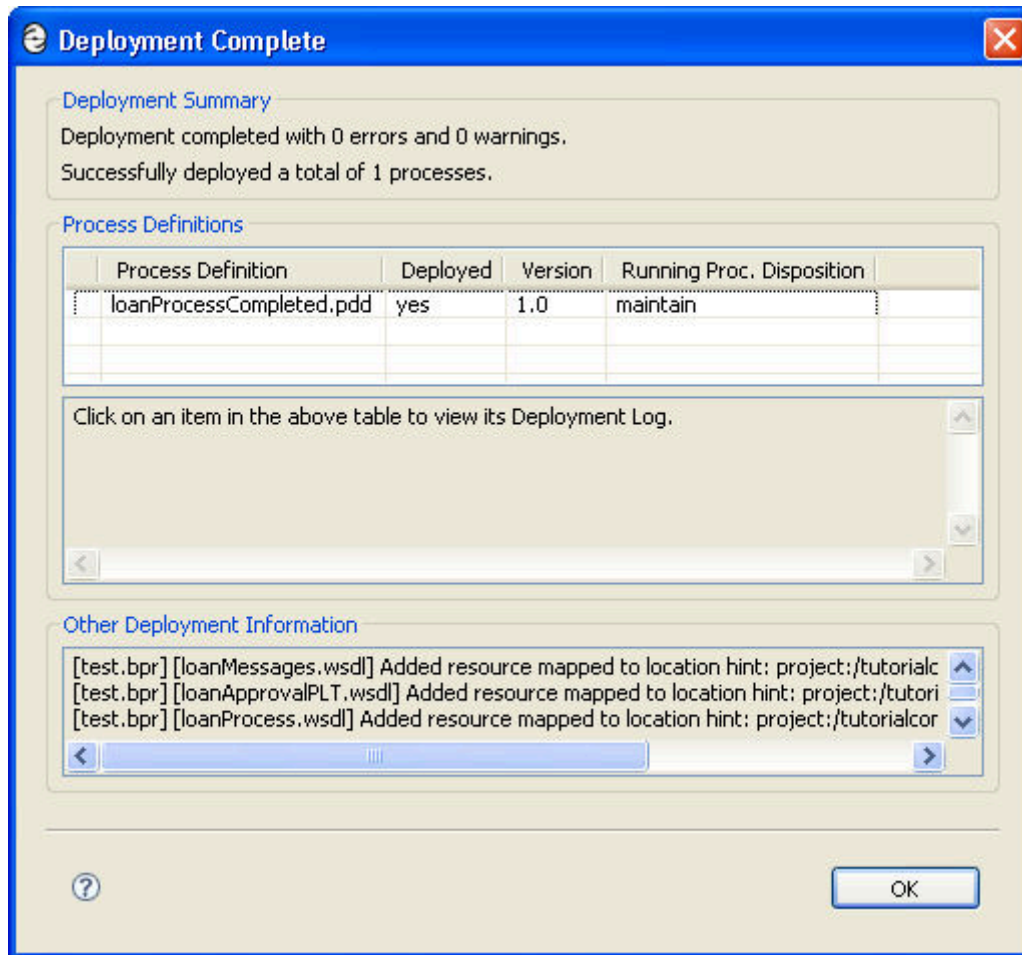
This dialog provides a quick look at the deployment results for your process files. Note the following:

- Each Process Developer process and resource successfully deployed has a version number. The version is provided by the deployment contribution.
- You can select a process definition to view a brief deployment log for it. For a complete deployment log, check the Contribution Detail page in the Process Console.
- WSDL files are not validated and are automatically deployed
- Deployment details for all resources appear in the lower part of this dialog

Starting the Server and Running a Process

After you deploy a process, described in "Creating and Deploying a Business Process Archive Contribution " on page 372, you can run the process by starting the server and sending the appropriate message to instantiate the process.

The Process Server that comes with Process Developer consists of the Process Server engine running under Apache Tomcat. Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies.

You must start the embedded engine before deploying your processes. When it starts, it scans for new, updated, and deleted `.bpr` files.

1. If you have not already done so, set up the server, as described in [“Setting Up the Embedded Process Server” on page 38](#).

2. Select the Servers view in the lower right of the workspace.

3. Select the **Start the Server** button, as shown in the example.
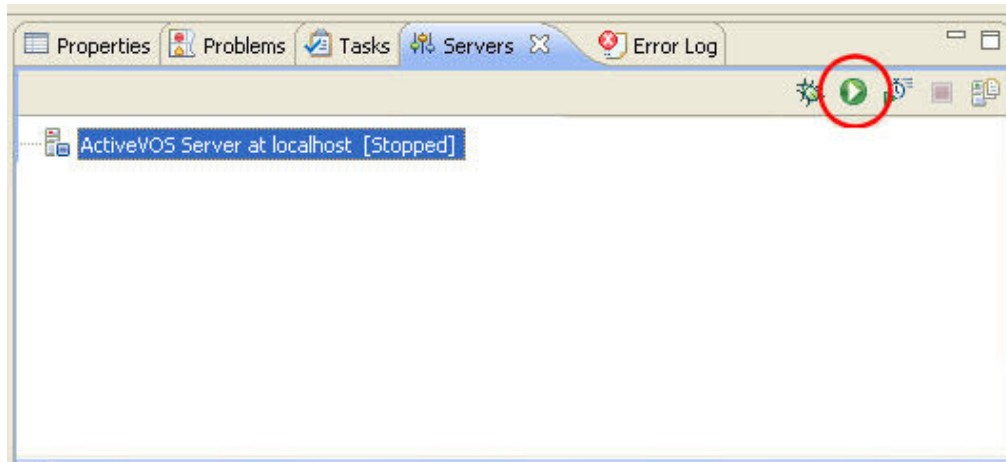
As the server starts up, you see start-up tasks scroll in the Console. Several files are deployed to the embedded server each time you start it. When the server is started, the State indicates this, as shown above.

You can use the Process Console to manage deployed and active processes. To display the Console, select the **Process Console** toolbar button and notice that the following URL appears in the browser address field:

```
http://localhost:8080/activevos
```

Change the port number if port 8080 is already in use on your computer.

For more information, select **Help** in the Console.

How a BPEL Process is Instantiated

A BPEL process is invoked just like a Web service. The process service is deployed with a URL of `http://[machinename:port]/active-bpel/services/[service name]`, where `[service name]` is the value of the service attribute of the my role element in the process deployment descriptor file.

The process is also deployed with a URL of `http://[machinename:port]/active-bpel/services/soap12/[service name]` so that clients sending a request to a process can do so with a SOAP 1.2 binding. The default binding is SOAP 1.1.

What is Process Versioning

Process Server manages process versioning. Versioning allows different versions for a given process to exist. Two deployments are considered to be different versions of the same process if they have the same target namespace and name in the BPEL file, but one deployment differs from the other in some way.

Process versioning allows you to control when processes become online (effective) and for how long. You can also control what happens to processes created by older versions when a new version comes online. While multiple versions of a process can exist concurrently, only the latest online version is capable of creating new process instances.

The latest version is in an *online* state. Other states include *online pending*, to describe versions that have an effective date in the future, *offline pending* to describe versions whose expiration date has arrived or has been set, and *offline* to describe expired versions that no longer have running process instances.

The process deployment descriptor provides selections for describing how a deployment is to be versioned. These selections are all optional and have default values as described below.

If desired, you can provide version information as described in "Creating a Process Deployment Descriptor File " on page 347.

The following example shows the syntax for version information in the `.pdd` file.

```
<version effectiveDate="2005-12-12T00:00:00-05:00"
    expirationDate="2007-12-12T00:00:00-05:00" id="1.5"
    runningProcessDisposition="maintain"/>
```

where:

- **online date** is the effective date the new version becomes the current version and all new process instances run against it. Depending on the disposition selected for running processes, some can continue to run until complete using the older version. The online date is an XML schema datetime value. Time is indicated as the midnight hour plus or minus the number of hours ahead of or behind Coordinated Universal Time (UTC) for the computer's time zone. In the example above, the computer time zone is eastern standard time, which is five hours behind UTC. For date/time details, refer to "Deadline and Duration Expressions " on page 257. If you do not provide an online date, it defaults to the date and time the process is deployed to the server.

- **offline date** is the date, beyond the online date, the current version expires. An offline version is not capable of creating new process instances. Once all of the running processes tied to an offline pending version complete then the version becomes offline. All process instances for the online version run to completion. The offline date is an XML schema datetime value. If you do not provide an offline date, the version does not go offline until you manually set it in the Process Console or until a newer version is deployed.

- **id** is the process version number in major.minor format. You do not need to provide a version number. The Process Server auto-increments new versions. The server increments a version number by dropping the minor value and adding 1 to the max number. For example, version 1.5 increments to version 2.0.

- **runningProcessDisposition** is the action the server takes on any other versions of the same process that currently have processes executing once this version's effective date arrives.
  If you specify *Migrate*, be sure to view the Server Log in the Process Console for possible warning messages describing incompatibilities between running processes and the new version they are running against. Refer to the Process Console Help for details.

Valid dispositions are:

- **Maintain**. Indicates that all process instances for the previous versions should run to completion. This is the default value.

- **Terminate**. Indicates that all process instances running under previous versions should terminate on the online date of the new version, regardless of whether or not the process instances are complete.

- **Migrate**. All running process instances created by previous versions will have their state information migrated to use the newly deployed process definition once its online date arrives. If there are incompatible changes between the versions, descriptive warning messages are generated in the Process Console Server Log. Refer to the Process Console Help for details.

C H A P T E R 2 7

BPEL Unit Testing

The follow topics discuss BPEL unit testing:

What is BPEL Unit Testing

Create individual B-unit test cases manually, or save each simulation path you run as a B-unit test file.

BPEL unit (B-unit) testing is a framework that tests BPEL processes in the Process Server engine. B-unit testing validates process execution of invokes, receives, and replies. You can use B-unit testing to perform automated regression tests on processes.

When a B-unit test runs, the following steps occur:

- The SendMessage command sends XML data to process. You provide this data as part of editing or generating a B-unit test.
- The response comes back from the process
- An assert expression determines what the response must have
- The result is compared to the assert

An effective test is to compare entire chunks of XML to see the parts that don't match.

You can create individual B-unit test cases manually, but it is recommended that you save each simulation path you run as a B-unit test file. You can then rerun files as needed as you refactor your process. Readily available B-unit tests make it easy for you to check whether a component of your process is still working properly.

You can design unit tests to produce test cases that cover all paths through a process so you can test various scenarios. You can save a group of files into a suite, or suite of suites, to rerun. You can run B-unit tests outside of Process Developer, as part of your automated build procedures.

For details, see:

- "Creating a BPEL Unit Test File" on page 384
- "Running a BPEL Unit Test in Process Developer" on page 385
- "Creating and Running a B-unit Ant Script" on page 386
- "Editing a B-unit File " on page 389
- "Debugging a B-unit Test " on page 394
- "Creating and Running BPEL Unit Test Suites " on page 394
- "Tips on Using Assertions " on page 396
- "Tips on Using Parameterized XSL for Input and Assert Data " on page 397
- "Tips on Providing Partner Link Data " on page 397

Creating a BPEL Unit Test File

Use one of the following techniques to create a B-unit test file:

- (Recommended) Simulate a BPEL process. In the Debug View toolbar, select **Save Simulation as B-unit Test Case**, as shown in the illustration.

- Right mouse click on a BPEL file and select **New > B-unit Test Case**. The B-unit Editor opens. This gives you a starting point for manually editing the file to provide data values and element attributes to test normal and faulting processes. For details, see "Editing a B-unit File " on page 389.
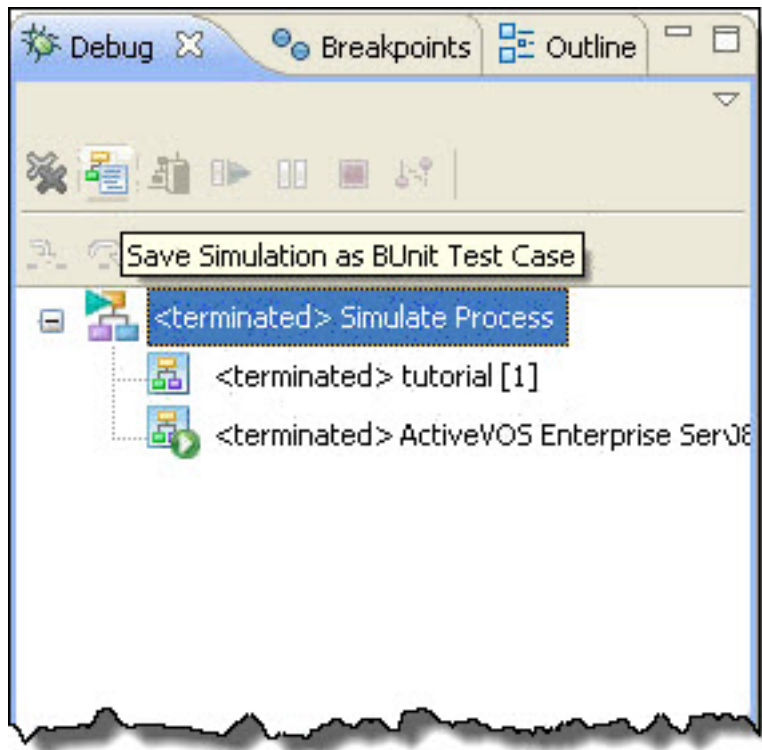
Tip: You can save a B-unit file into the test folder within an orchestration project. Regardless of where you save the file, after you save it, it opens in the B-unit editor.

See also "Running a BPEL Unit Test in Process Developer" on page 385and "Creating and Running BPEL Unit Test Suites " on page 394

Running a BPEL Unit Test in Process Developer

You can run B-unit tests from an Ant script outside of Process Developer or from the B-unit View within Process Developer. For details on command line execution, see "Creating and Running a B-unit Ant Script" on page 386.

You can run a B-unit test in Process Developer by using the Run As command from several locations, including right-mouse clicking on a B-unit test or test suite in the Project Explorer and selecting **Run As > B-unit**.

The B-unit view opens and runs the test. The following examples show a B-unit with a completed test and a failed test:
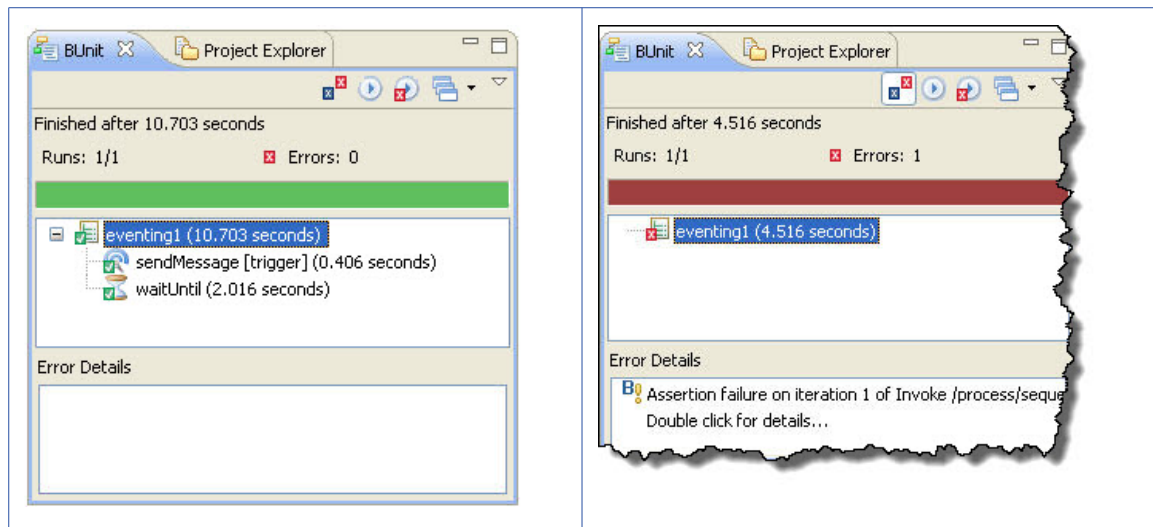
You can also run one test from a B-unit suite, as described in "Creating and Running BPEL Unit Test Suites " on page 394.

When you run a test the following events occur:

- The B-unit view opens to show the progress of the test.
- The Console shows the output of the test. By default, it includes the trace events and the process execution. For details of these events, see "Editing a B-unit File " on page 389.
- If a test fails, a failure message displays in the B-unit view. For certain test errors, you can double-click on the test that failed, and a Text Compare Editor opens to show actual and expected results.
- In the Text Compare Editor, the differences between the expected and actual results are shown. You can use the **Toggle On/Off Normalized View** to simplify looking for file differences. You can also set a color preference for the Text Compare Editor, as described in "Colors and Fonts Preferences" on page 75.

When the test completes, you can click on an event in the B-unit view to go directly to that section of the B-unit test in the B-unit editor.

In the B-unit view, use the toolbar icons:

- **Show Failures**. Select this to filter the view to show only tests that fail.
- **Rerun Test**. Once you run one test, you can rerun it from the B-unit view.
- **Rerun All Failures**. Rerun failed tests from the B-unit view.
- **History**. Remembers the last 15 unit tests run. You can select a test from the list based on the name of the B-unit file.

Creating and Running a B-unit Ant Script

Create individual B-unit test cases manually, or save each simulation path you run as a B-unit test file.

If you want to include B-unit testing in an automated build environment, you can create an Ant script to run all the B-unit tests and suites from multiple orchestration projects outside of Process Developer, on the

command line. This capability is similar to using the command line BPRD script to deploy a BPR archive to the server.

Prerequisite

You must install the **B-Unit Ant Runtime** component of the Process Developer before you can run the Ant script. The installation creates a directory containing the runtime files required to run the B-unit Ant script from the command line, outside of Process Developer. You do not need to install Ant separately: the Apache Ant application is installed as part of Process Developer.

Note: Version 1.7.0 of Ant (or above) is required.

Creating a B-unit Ant Script

Use the following procedure to create a B-unit Ant script:

1. Ensure your orchestration projects contains one or more B-unit tests or suites, as described in <u>"Creating a BPEL Unit Test File" on page 384</u>.

2. Select one B-unit test or suite and right mouse-click to select **New > BUnit Ant Script**.

3. In the Export B-unit Ant Script wizard, select a folder, such as the *test* folder, and enter a filename. Select Next.

4. Select the B-unit tests and suites to include. You can select multiple tests from multiple projects. Also, after you create the script, you can edit it to add more folders of tests.

5. Select the B-unit options as desired:

    *   **Enable Tracing**. Controls console logging output for B-unit-specific execution commands.

    *   **Enable Process Logging**. Controls the amount of process execution data to log. The default is *none*. Enabling the checkbox turns on *execution* level. If desired, you can manually edit the Ant script to enable *execution-data* (execution with data) or *full* logging.

6. Select **Finish** to generate the `filename`.ant.xml file.

The ant script opens in an editor for you to view or edit. Follow the instructions in the ant script to set the B-unit runtime classpath if you wish to execute this script outside of Process Developer.

You can create a code coverage report when you run a B-unit ant script on the command line. For details, see <u>"Generating a Code Coverage Report " on page 388</u>.

Notes on Modifying a B-Unit Ant Script

Within the ant script, you can override properties in the individual B-unit files.

*   You can add a file to override the default-B-unit-engine config if needed, such as `<config file="engine-configuration.xml" />`

*   Nested element `<engineProperties>` accepts a file relative or absolute path to a properties file with engine *config path=value* pairs in the format required by java.util.Properties. The file attribute is optional and instead nested `<engineProperty name="…" value="…"/>` elements can be used to set engine properties. If both the file attribute and the engineProperty elements are used, the elements will override the values in the properties file if the same config path is specified twice.

*   Examples of multiple engine properties:

    - `<engineProperty name="Logging" value="urn:ae:execution" />`

    - `<engineProperty name="CustomFunctions/MyCustomFunction"`
      `value="com.test.my.custom.Function" />`

Adding Multiple B-unit and B-suite to an Ant Script

Ant provides many ways to specify resource collections, such as a group of B-unit and B-suite files. One example is the following.

```
<path>
    <fileset dir="${test.dir}" includes="**/*.bunit"/>
</path>
```

For examples, refer to the *resource collections* section of the Ant 1.7 (or later) documentation.

Running a B-Unit Ant Script

You can run the `filename.ant.xml` file from within Process Developer if desired. Right-mouse click on the file and select **Run As > Ant Build**. In the Console you can view trace and log details.

To run the Ant script on the command line, do the following:

1. Open a command window.

2. Navigate to the file system folder where the ant script is located.

3. At the command line prompt, type `ant -f filename.ant.xml`

Generating a Code Coverage Report

Using a B-unit ant script, you can generate a code coverage report. The report tracks execution paths and display which activities were executed, expressed as percentage of activities executed at least once, in relation to all process's activities.

To generate a code coverage report:

1. Create a B-unit ant script as described in ["Creating and Running a B-unit Ant Script" on page 386](#).

2. In the `<target>` named `instrument`, remove the comments and fill in a resource collection of bpel files. The following example show a simple resource collection:

```
<target depends="init" name="instrument">
<!-- optional
<coverage activitycoverage="true" linkcoverage="true" />
-->
 <bunit-instrument>
   <filelist>
        <file name="../bpel/myFile.bpel"/>
     </filelist>
</bunit-instrument>
```

By default, a code coverage report is generated for activities. If you want to include links, add the optional `<coverage>` element.

3. In the target named `test`, uncomment the `<engine properties>` section.

4. In the target named `coverage report`, do the following:

   - Provide a destination directory for your report.

   - Provide a report type. The types are HTML (default), XML (raw XML used to create HTML) or CONSOLE (writes summary results to the Ant Logger that you specify on the command line)

   - If you create a HTML report, you can add an xsltPath attribute.

   - Rename the filePrefix, if desired. The Ant project name is the default filePrefix. When you generate a report, the date and time are appended to the prefix.

5. Run the B-unit ant script on the command line to generate the report. For example, in a command window, type `ant -f filename.ant.xml`.

What the Code Coverage Report Shows

- Activity Coverage (default coverage): the total number of activities executed at least once over the total number of activities in the process.

- Link Coverage (optional): the total number of activities that evaluated to true plus those that evaluated to false over the 2x total number of links with conditions.

    `(totalTrue + totalFalse) / (totalLinksWithConditions *2)`.

  This ensures that the unit tests cover the execution of links completely because each link with a condition can evaluate to true or false.

Adding Code Coverage to an Existing B-Unit Ant Script

If you have an existing B-unit ant script, you can add code coverage to it just by putting in the correct ant tasks in the B-unit ant script.

- Register the following typedefs in your Ant script and note that the attribute *loaderref* is set to *bunit*. This attribute must be set in order for all the Ant tasks to use the same classloader. The value of the attribute is unimportant as long as the value for all three typedefs match:

```
<typedef classname="com.activee.ant.bunit.tasks.AeBUnitTask"
        loaderref="bunit" classpathref="bunit.classpath" name="bunit"/>
<typedef classname="com.activee.ant.bunit.coverage.AeBUnitInstrumentTask"
        loaderref="bunit" classpathref="bunit.classpath" name="bunit-instrument"/>
<typedef classname="com.activee.ant.bunit.coverage.AeBUnitCoverageReportTask"
        loaderref="bunit" classpathref="bunit.classpath" name="bunit-report"/>
```

- Instrument your processes that are part of code coverage:

```
<bunit-instrument>
        <coverage activitycoverage="true" linkcoverage="true" />
        <filelist>
            <file name="../bpel/myBEPL.bpel"/>
        </filelist>
</bunit-instrument>
```

- Add the following engine property to the B-unit task that will execute the B-unit tests as follows:

```
<bunit processLogging="urn:ae:full" trace="on">
        <engineProperties>
           <engineProperty name="CustomManagers/AeBUnitCoverageManager/Class"
            value="com.activee.rt.bunit.coverage.AeBUnitCoverageManager"/>
        </engineProperties>
         <filelist>
            <file name="CancelTO.bunit"/>
            <file name="CreateTO.bunit"/>
            <file name="SubmitPO_UNI.bunit"/>
            <file name="SubmitUNIPO_AIFallOut.bunit"/>
         </filelist>
</bunit>
```

- Finally, you will need to output the report using the bunit-report Ant task:

```
<bunit-report reportType="HTML"
        destdir="/Users/User1/Downloads/bunit/html" />
```

For details on the parameters you can use for the B-unit report task (filePrefix, xsltPath, destDir, and reportType, see `Step Four` in the above procedure.

Editing a B-unit File

You can design your own B-unit tests with the help of the B-unit editor. Double-click a B-unit file in Project Explorer to open the B-unit editor.

The B-unit editor displays the Design tab, containing an Outline pane and a Details pane. The Outline displays different sections of a B-unit file that you can select to edit in the Details pane.

The following is a description of each node in the B-unit Outline. For more details, select the node name.

- "BPEL Unit (Root) " on page 390
  Specifies the BPEL, WSDL, Schema and other resources used in the test. The resources provide data that creates the set of deployed processes and resources for the Process Server engine. When the B-unit test is started, all of these resources are deployed to the engine much as they would be when the standard engine starts up. The standard static analysis rules for Informatica Business Process Developer are run against each BPEL process identified in the B-unit to ensure that there are no errors in the BPEL or environment.

  The Trace option controls console logging output. It is enabled by default and can be useful for tracking down assertion failures if the B-unit view does not provide adequate debugging information.

- "Extensions and Extension Activities " on page 391
  Extensions specify Logical People Groups that are WS-HT (Human Task) Extensions. Extension Activities specify People activities that are BPEL for People Extension activities.

- "Invokes " on page 392
  Specifies invoke activities.

- "Alarms " on page 392
  Specifies Wait activities and onAlarm handlers for picks and event handlers. The alarms and invokes elements contain information about each alarm and invoke within the process that is expected to execute during the test. These elements should be thought of as re-active in the sense that they react to the execution of an alarm or invoke within the process. They do not cause an alarm or invoke to execute, but rather wait for the engine to signal that an alarm or invoke has executed and then match that activity within the process to a declaration within the B-unit. Once matched, the B-unit declaration will provide assertions, simulated response/fault/alarm.

- "Commands " on page 393
  Sends messages to the B-unit test engine to execute the test logic. The commands element contains all of the commands that are sent into the engine when the B-unit test executes. These elements should be thought of as active in the sense that they are actions taken by B-unit when it starts the test. Each command is executed in sequence. If the command has an *asynch="true"* attribute then this command is executed on a separate thread. The B-unit test is considered completed when all commands complete. The most common command is the sendMessage command and is the only required command for each B-unit.

BPEL Unit (Root)

Open a B-unit file in the B-unit editor, and select BPEL Unit from the Outline view.

The detail view displays the BPEL, WSDL and other resources used in the B-unit test.

To add resources, specify name, location and type URI. If you are using an XQuery module, the type URI is:

```
http://modules.active-endpoints.com/2009/07/xquery
```

At the bottom of the panel is the **Trace** selection. You can turn trace on or off to control console logging output.

The child elements of BPEL Unit are:

- Extensions

- Extension Activities

- Invokes

- Commands

- Alarms

To add a child element, right-mouse click on BPEL Unit, and select **Add**.

See also, "Editing a B-unit File " on page 389.

Extensions and Extension Activities

Open a B-unit file in the B-unit editor, and select **Extensions** or **Extension Activities** from the Outline view. To add an extension, right-mouse click on BPEL Unit.

Extensions

Extensions specify context information required to run a B-unit that are typically defined in a process deployment description. Currently, extensions specify Logical People Groups that are WS-HT Extensions, and can be used within People activities.

You can define the value for a Logical People Group exactly as you define it in the People tab of the Process Deployment Descriptor Editor. For details about static, dynamic and People Query values.

Extension Activities

Extension Activities refer to People activities. You can assert what goes into a People activity and also to specify a response from the People activity if it includes a task (and not a notification).

You can define a People activity for a B-unit test as follows:

1. Select or add an Extension Activity (a People activity). If this activity is within a loop (for example, a For Each), you can specify a Count for the expected number of times to execute the activity.

2. Add at least one iteration that is the Default iteration. This is the default for how each iteration should run.

3. For each iteration, add the Response type that is the outcome of a task:

   - **Completed.** Specify the Output data returned from the task. The data can be Imported or created Inline. By default, the data is set to be imported. You can set a preference for inline, if desired. If the data is imported, and if desired, you can add your own XSL files to transform the XML data returned from the task. This allows you to reuse the commonality within the data for different iterations while changing only a few elements of the data. See "Tips on Using Parameterized XSL for Input and Assert Data " on page 397.

     If desired, specify the context data; that is, the names of task initiator, actual task owner, and other assignments for the task.

   - **Failed**. Select a Fault to return.

   - **Expired**. The task reached its expiration date, if the expiration is set.

   - **Skipped**. The task was skipped, if the Skippable option is selected in the task.

4. **Add Asserts**

   - **Assert Match**. Specify a Label that is documentation for this Assert, message part and query for the assert, and a pattern to match on. An example might be to match on data returned from one of the B4P custom functions.

   - **Assert Equal**s. Specify the data that should match the data returned.

   - **Assert Task**. Asserts the task details (for example, subject, presentation, and escalations).

See also, "Editing a B-unit File " on page 389 and "Tips on Using Parameterized XSL for Input and Assert Data " on page 397.

Invokes

Open a B-unit file in the B-unit editor, and select Invokes from the Outline view. To add an invoke, right-mouse click on BPEL Unit.

A common use case for an invoke is to provide a simulated response for a service that your process is invoking. When an invoke activity within the process executes, the B-unit engine looks for a matching invoke element within the B-unit file. If found, this invoke element provides the ability to do assertions on the input data passed to the invoke as well as provide a message that simulates the response from the service invoke.

B-unit invoke elements can be configured for one-way and two-way service invokes. Both styles of invokes can have assertions on the input data but only two-way service invokes are allowed to have a response.

You can define an invoke for a B-unit test as follows:

1. Select or add an Invoke. Add an Invoke by right-mouse clicking on BPEL Unit.

2. Specify the Name or **L**ocation Path of the invoke. If this activity is within a loop (for example, a For Each), you can specify a Count for the expected number of times to execute the activity. Specify a Delay if you want to delay before returning the response. The delay allows for better handling of onEvent and onAlarm events.

3. Add a default iteration.

4. Add the Response type expected:

   - **Simulated Respons**e. Specify the Output data.

   - **Subprocess**. The invoke is a BPEL process, deployed as a subprocess, making the main process eligible for termination and compensation handling.

   - **Process**. The invoke is a BPEL process, deployed as a "process" invoke handler.

   - **Simulated Fault**. The invoke returns a fault.

5. Add Asserts.

   - **Assert Match**. Specify a Label that is documentation for this Assert, message part and query for the assert, and a Pattern to match on.

   - **Assert Equals**. Specify the input data passed to the invoke.

About Message Part Data

Some information about a message and its associated parts are generated automatically in the B-unit editor when you perform certain actions including:

- Fill in values for process name, partner link and operation for a Send Message.

- Select Simulated Response for an invoke assertion (or extension assertion).

- Select a fault name for a Simulated Fault.

See also, "Editing a B-unit File " on page 389, "Tips on Using Assertions " on page 396, and "Tips on Using Parameterized XSL for Input and Assert Data " on page 397.

Alarms

Open a B-unit file in the B-unit editor, and select Alarms from the Outline view. To add an alarm, right-mouse click on BPEL Unit.

Alarms refer to the wait activity and the onAlarm events in picks and event handlers. You can assert on an alarm deadline or duration.

You can define an alarm for a B-unit test as follows:

1. Select or add an alarm. Add an alarm by right-mouse clicking on BPEL Unit.

2. Specify the Process and Location Path of the alarm-based activity.

    • Add the Simulated Duration of the alarm. This is the time the B-unit test will wait for the alarm to be triggered.

    • If this activity is within a loop (for example, a For Each), you can specify a Count for the expected number of times to execute the activity.

3. Add a default alarm, and add the duration for the B-unit test to use.

4. Add an Assert.

    • Assert Deadline. Specify the actual deadline, converted to a duration, for the B-unit test. Specify a Precision value that is the period of time within which the deadline should fire. This value is specified in seconds.

    • Assert Duration. Same as deadline.

5. Add an assert iteration, if desired. If the alarm-based activity is within a loop, add an Index for this iteration. The index is the *nth* iteration within the count, specified for the alarm.

    See also "Editing a B-unit File " on page 389 and "Tips on Using Assertions " on page 396

Commands

Open a B-unit file in the B-unit editor, and select Commands from the *Outline* view. To add a command, right-mouse click on BPEL Unit.

Commands send messages to the B-unit test engine to execute the test logic.

The commands you can add are as follows:

Command	Description
Send Message	Sends WSDL messages to the test engine. Select an Operation for the specified partner link. Delivery Options. - **Async**. Controls whether or not two receives can execute in a row, without waiting for the reply associated with the first receive to complete - **Delay**. Wait value that the B-unit waits before starting to **check** the Timeout. - Timeout. Sets a value for the how long the test engine should wait for process completion. This setting ensures that the invokes complete with the correct count. - **Message Context**. Optionally specify the meta-data for the incoming message, including the authenticated principal, and other message header data.
Wait Until	Checks for the execution completion of the process. - **Delay**. Wait value that the B-unit waits before starting to check the Timeout. - **Timeout**. Sets a value for the how long the test engine should wait for process completion. This setting ensures that the invokes complete with the correct count. - **Fail on Timeout**. Enable this check box to fail the test if the process does not complete within the timeout value.

Command	Description
Terminate Process	Terminates a process instance for one of the processes in the B-unit test, if you are running more than one process.
Assert Process State	Asserts the execution state of the process that you specify. - **Process State**. Running, suspended, complete, faulted, final. Select final to assert completed and faulted states. - **Process Id**. Used only if you are running the test on multiple BPEL files. - **Delay**. Wait value that the B-unit waits before starting to check the Timeout. - **Wait for Async**. If you have Async selected in a Send Message command for the same process, this property allows you to wait to assert the process state until all asynchronous activities have completed.

See also, "Editing a B-unit File " on page 389.

Debugging a B-unit Test

If your B-unit test fails, you can right-click on the test name in the B-unit view, and select **Debug**. The Debug view opens and your test is run. The Console displays the logging output from the test.

In Debug mode, you can set breakpoints in the BPEL file and debug it, just like remote debugging.

Creating and Running BPEL Unit Test Suites

You can multi-select B-unit files in the Project Explorer and make a suite of tests.

For example, select two or more `.bunit` files from your test folder within an orchestration project. Right-mouse click and select **New > BSuite Test Suite**. The BSuite dialog opens, as shown:
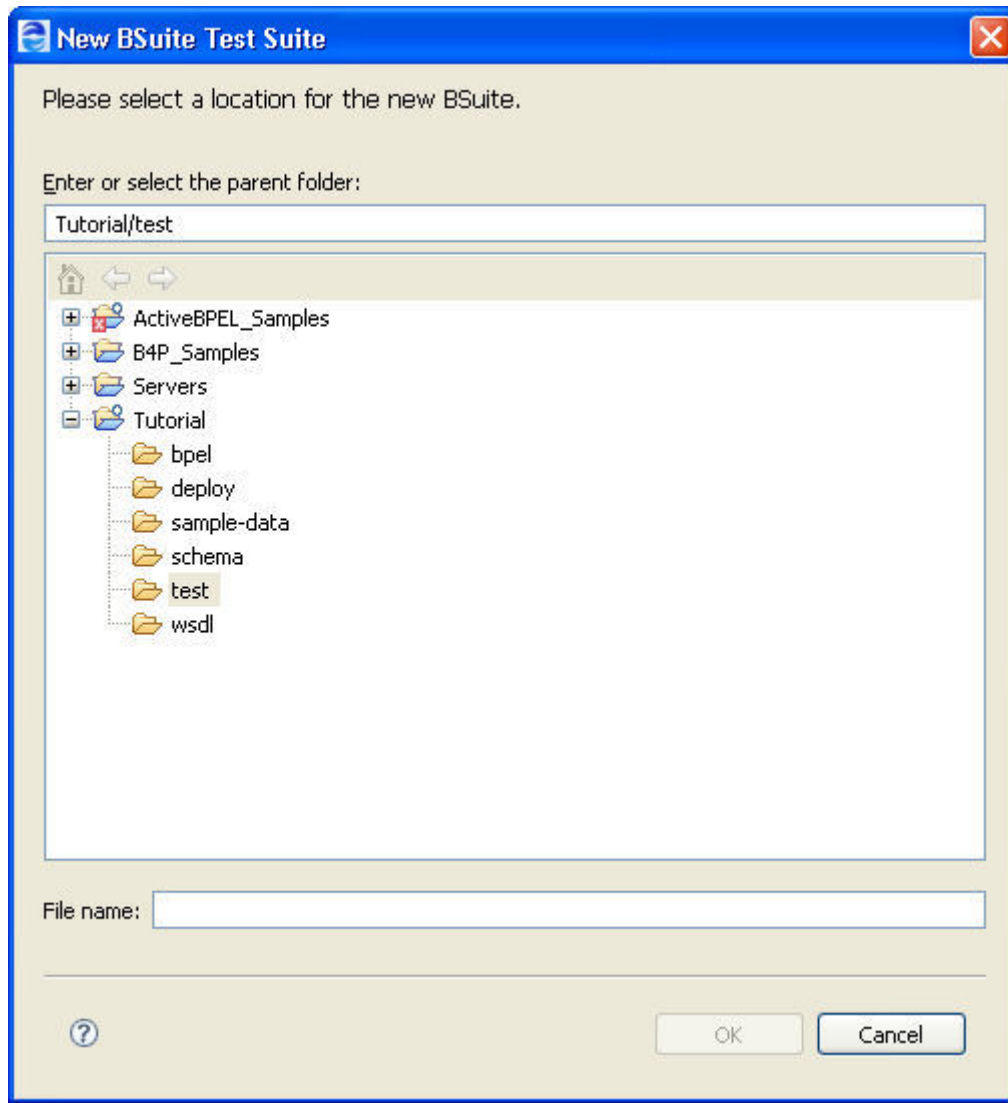
Type in a name for the BSuite. The `.bsuite` extension is automatically added.

All the tests in the suite run.

You can add BSuites to a BSuite, as shown:
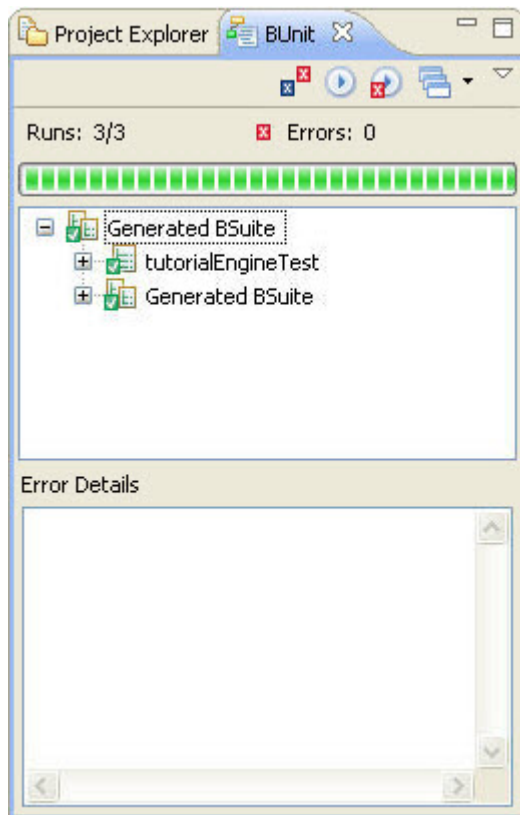
Tips on Using Assertions

A key component of the B-unit testing framework is the ability to perform assertions. An *assertion* describes the expected correctness of the state of a unit of code when it runs. Assertions include:

- Input data for invoke activities
- Replies from synchronous messages sent into the engine via the B-unit Send Message command
- Values for alarm deadlines and durations

Assertions contain expected data and run against data from the process execution. The expected data for the assertion can be a text node, an element, or an imported element. The assertion can run against the entire message data or can use an XPath query to select a portion of the message data.

The B-unit engine validates messages for receives/replies/invokes. As such, it is unnecessary to do assertions on a full message. Instead, you can write assertions that focus on the portion of a message that tests logic within your process. This type of focused assertion makes your unit tests more readable and reduces the amount of work necessary if the message definitions ever change. For example, if you want to assert that the value of `rm:ProductGroupId` is `100`, the following query provides you with the most flexibility:

```
<abu:assertEquals part="p"
    query="//rm:ProductGroupId/text()">100</abu:assertEquals>
```

In this example, note the use of the descendant axis in the query. This selects the `ProductGroupId` element anywhere in the message. While this lack of specificity is not always a good idea, keep in mind that the B-unit engine is already validating the structure of a message so if the element were misplaced, this is reported in schema validation.

Here are some good uses of assertions:

- Asserting the input for an invoke when that input was produced by one or more complex data mappings within the process
- Assertion proves that a specific path was taken within the process in the case where there is conditional logic in the process in the form of transition conditions or if/else activities

It is probably not worth doing an assertion on something you would get with schema validation or with a large literal assertion when there's only a small part of the message that is worth asserting.

Tips on Using Parameterized XSL for Input and Assert Data

You can build efficiency into your B-unit tests in cases where you want to assert a small difference in the same data across multiple tests. The efficiency comes in the form of parameterized XSL.

Consider the following example.

In an XML data file, there is the following element:

```
<ns:status>CREATED</ns:status>
```

Instead of creating different XML files, each with a different status value, you can write an XSL file such as:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ns="http://schemas.active-endpoints.com/b4p
      /wshumantask/2007/10/aeb4p-task-rt.xsd">
    <xsl:import href="common.xsl"/>
    <xsl:param name="state"/>
    <xsl:template match="ns:status/text()" priority="2">
        <xsl:value-of select="$state" />
    </xsl:template>
</xsl:stylesheet>
```

In the Source view of your B-unit file, you can then add a parameter, such as:

```
<abu:part href="../data/getTaskInstanceResponse-minimal.xml"
    name="taskInstance"
    xsl="../xsl/change-task-instance-state.xsl">
    <abu:params>
        <abu:param name="state" value="COMPLETED"/>
    </abu:params>
```

For each B-unit test, you use the same imported data and the same XSL file. You need only change the parameter value to get a different result.

Tips on Providing Partner Link Data

Some processes require the configuration of partner link data as part of the test. Partner links can be specified along with `wsa:EndpointReference` information for their partnerRole or service names for their myRole. This is only necessary if the process makes use of some data within a partner link during some calculation.

To add partner link data to a B-unit test:

1. In Outline view, select the Partner Links node, and select a partner link.
2. In the Properties view, select the All tab, and select the Simulation category.
3. Select the appropriate partner link, My Role, or Partner Role.
4. Select the Dialog button at the end of the row to open the **Partner Link Data** dialog.
5. Type in an Endpoint Reference definition.
6. Simulate the process and save the simulation as a B-unit test.

The following example shows the B-unit element that is added to your test.

```
<!-- Controls deployment of partner links to the test engine. -->
<abu:partnerLinks>
  <abu:partnerLink xmlns:ns="http://docs.active-
    endpoints.com/sample
    /bpel/loanprocess/2008/02/loanProcessCompleted.bpel"
    name="LoanApproval"
     processName="ns:loanProcessCompleted">
      <abu:partnerRole endpointReference="static">
       <wsa:EndpointReference
         xmlns:s="http://docs.active-endpoints.com/
           sample/wsdl/riskAssessment/2008/
           02/riskAssessment.wsdl"
           xmlns:wsa="http://www.w3.org/2005
           /08/addressing">
           <wsa:Address>urn:x-vos:loancompany:
             RiskAssessmentServiceNEW
           </wsa:Address>
             <wsa:ServiceName
                PortName="RiskAssessmentServicePort">
                      s:RiskAssessmentServiceNEW
             </wsa:ServiceName>
        </wsa:EndpointReference>
      </abu:partnerRole>
   </abu:partnerLink>
</abu:partnerLinks>
```

Example B-unit File

Here is a B-unit source template, showing most of the sections of a B-unit file.

```
<?xml version="1.0" encoding="UTF-8"?>
<abu:bpelUnit xmlns:abu="http://schemas.active-endpoints.com
    /activebpelunit/2008/10/activebpelunit.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <abu:trace>on</abu:trace>
  <abu:engineProperties>
     <abu:documentation>Controls engine configuration
        options, including process logging.
     </abu:documentation>
     <abu:engineProperty path="Logging"
         value="urn:ae:execution"/>
  </abu:engineProperties>
  <abu:bpels>
     <abu:documentation>Controls deployment of the BPEL
           process to the test engine.</abu:documentation>
     <abu:bpel location="../bpel
           /loanProcessHumanCompleted.bpel"/>
  </abu:bpels>
  <abu:wsdls>
     <abu:documentation>Controls deployment of the WSDL files
        to the test engine.</abu:documentation>
     <abu:wsdl location="../wsdl/loanProcess.wsdl"/>
  </abu:wsdls>
  <abu:schemas>
     <abu:documentation>Controls deployment of the XML Schema
           files to the test engine.</abu:documentation>
     <abu:schema location="../schema/loanRequest.xsd"/>
  </abu:schemas>
  <abu:resources>
     <abu:documentation>Controls deployment of resource
        mappings to the test engine.</abu:documentation>
     <abu:resource location="../xsl-renderings/
        loan_taskdetail.xsl" locationHint=
        "project:/humantaskcompleted/xsl-renderings/
```

```
                loan_taskdetail.xsl"
                type="http://www.w3.org/1999/XSL/Transform"/>
        <abu:resource location="../xsl-renderings/
            loan_param2commands.xsl" locationHint=
                "project:/humantaskcompleted/xsl-renderings/
                 loan_param2commands.xsl"
                type="http://www.w3.org/1999/XSL/Transform"/>
    </abu:resources>
    <abu:invokes>
        <abu:documentation>Defines the simulated output and
          assertions to perform on invoke activities
        </abu:documentation>
        <abu:invoke count="1" name="NotifyCRMOfExpiration">
            <abu:invokeIteration index="0"/>
        </abu:invoke>
    </abu:invokes>
    <abu:commands>
        <abu:documentation>Sends messages to the B-unit
            test engine
            to execute the test logic.</abu:documentation>
        <abu:sendMessage async="true"
            operation="request" partnerLink="LoanProcess">
        </abu:sendMessage>
        <!-- Tells the test to wait before making final assertions -->
        <abu:waitUntil xmlns:
          ns="http://docs.active-endpoints.com/sample/bpel
          /loanprocess/2008/02/loanProcessHumanCompleted.bpel"
            processName="ns:loanProcessHumanCompleted"
            timeout="5000"/>
    </abu:commands>
</abu:bpelUnit>
```

CHAPTER 28

Creating POJO and XQuery Custom Functions

In Process Developer you can write custom functions, either as POJO (Plain Old Java Object) or XQuery functions. Built-in functionality makes creating and using custom functions in the Expression Builders easy.

Custom Functions Overview

You can create and use custom functions in BPEL processes. A custom function is a POJO that implements a function context interface from Process Developer library. It also contains Process Developer annotations to register and display custom functions in the Functions list of the Expression and Query Builders.

You can create and use custom functions in one or more projects. A custom function is deployed as a project resource within a deployment contribution. Custom functions can be in the same project as your BPEL files or in referenced project. Any JAR files that support your custom functions must be in the classpath of your project.

For an easy way to implement the function context interface, create a new Java enabled orchestration project, described in "Orchestration Project Templates" on page 30. The required Process Developer library is automatically added to your project, as shown in the illustration:
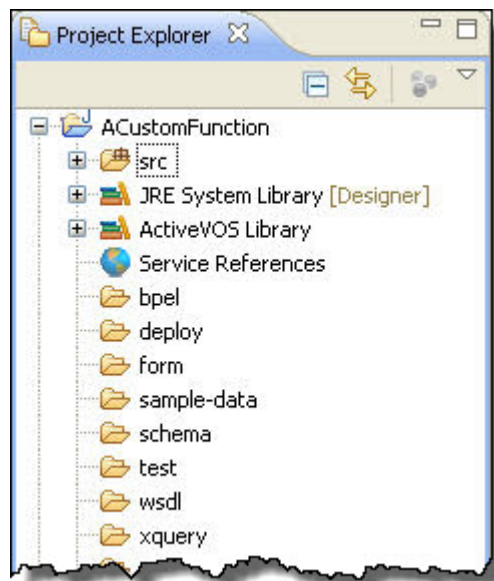
If you have an existing Java-enabled orchestration project, add the Process Developer library as follows:

1. Right-mouse on the project name and select **Properties**.

2. Select **Java Build Path**.

3. From the Libraries tab, select **Add Library**.

4. Select Process Developer Library as shown in the example.
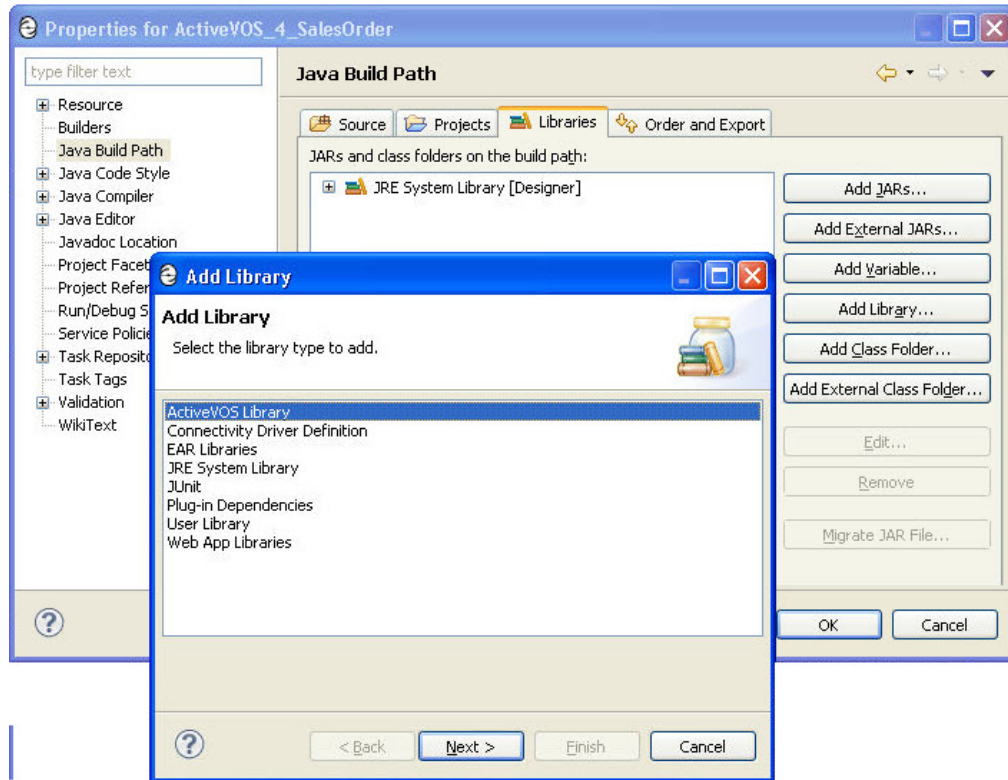
See also "Implementing the Function Context and Adding Annotations " on page 401.

Implementing the Function Context and Adding Annotations

The Process Server uses a Java framework that supports custom function extensions. This framework is a set of Java interfaces and classes made available for your use. `IAeFunction` and `IAeFunctionContext` are implemented to create the objects that are accessible to the engine at run-time. `AeFunctionCallException` and `AeUnresolvableException` are used to signal fault conditions. These types are resolved using the `org.activebpel.rt.bpel` package in `ae_rtbpel.jar`. Additionally, `AeUnresolvableException` extends AeException provided in `ae_rt.jar`.

You can use custom functions in any expression language.

Implementing: Step 1

Your first task is to create a function context that is a new class that implements the `IAeFunctionContext` interface.

This class exposes one public method: `getFunction(String)`. It returns an object of a type that implements the IAeFunction interface. The method uses the String argument to indicate the local name of the function and then select that object.You can implement multiple custom functions via one function context class. `getFunction()` throws an AeUnresolvableException if it is unable to locate the desired function by name.

Annotations

You add the `@AeFunctionContext` annotation to annotate the class at class level.

Add the `@AeFunctionUnit` annotation at the method level, for each function unit that you want to display in the Expression Builder function list.

The following example shows the function context:

```
@AeFunctionContext(name="myContext", namespace="myns")
 public class CustomContext implements IAeFunctionContext
 {
    @AeFunctionUnit(
     prefix = "ld",
     display = "MyLDAP",
     hoverText = "My LDAP Functions",
     functions = {
        @AeFunction(
          syntax = "${prefix}:getOrgUnit(${caret})",
          display = "getOrgUnit(id)",
          hoverText = "Returns OU details"),
        @AeFunction(
          syntax = "${prefix}:getSiblings(${caret})",
          display = "getSiblings(id)",
          hoverText = "Returns siblings nodes"),
    }
 )
  public IAeFunction getFunction(String aFunctionName)
  throws AeUnresolvableException
    {
    }
  }
```

where:

- `@AeFunctionContext name` is the display name of the function context in the Process Console Catalog

- `@AeFunctionUnit` represents a group of functions

  - `prefix` is used for all functions in the unit

  - `display` is the unit name in the Expression Builder function list

  - `hoverText` is the unit's hover help

  - `functions` is the list of functions

- `@AeFunction` represents the function

  - `syntax` is the syntax of the function added to the Expression text box when the function is double-clicked. Note that the syntax includes the `${prefix}` and the `${caret}`. The cursor appears in the column set by the caret token.

  - `display` is the function name in the Expression Builder function list

  - `hoverText` is the function's hover help

Implementing: Step 2

Your second task is to create the actual custom functions, each of which implements the IAeFunction interface. These also expose one single public method: `call(IAeFunctionExecutionContext, List)`; it performs the function's task that you create. The function's argument list (if any) is passed to the call method using the `List` argument. This function may optionally use the function execution

`IAeFunctionExecutionContext` as needed. Individual custom functions should throw `AeFunctionCallException`, constructed to contain the root exception message, if they encounter a problem that prevents them from finishing their task.

Implementing: Step3

Once you have created your function context and function classes, and tested the classes that do the actual custom function work, you can create a contribution. All the function contexts in the project are automatically packaged when you create a deployment contribution.

The custom functions are available only to the BPEL files in the contribution or in a contribution in another project that references this project. Note that the custom functions are listed on the Catalog Resources page of the Export Business Process Archive, either as project resource dependencies, referenced dependencies, or if they are not referenced, they are listed as additional resources. For details, see .

Sample Custom Function

The following shows a sample custom function:

```
package com.acme.functions;
import java.util.List;
import org.activebpel.rt.bpel.function.AeFunction;
import org.activebpel.rt.bpel.function.AeFunctionCallException;
import org.activebpel.rt.bpel.function.AeFunctionContext;
import org.activebpel.rt.bpel.function.AeFunctionUnit;
import org.activebpel.rt.bpel.function.AeUnresolvableException;
import org.activebpel.rt.bpel.function.IAeFunction;
import org.activebpel.rt.bpel.function.IAeFunctionContext;
import org.activebpel.rt.bpel.function.IAeFunctionExecutionContext;
@AeFunctionContext(name = "AcmeContext", namespace = "http://acme.com/functions")
public class GenericFunctionContext implements IAeFunctionContext {
    private IAeFunction echoFunction = new EchoFunction();
    private IAeFunction companyNameFunction = new CompanyNameFunction();
    @Override
    @AeFunctionUnit(prefix = "ac", display = "ACME",
                hoverText = "Acme company functions", functions = {
            @AeFunction(syntax = "${prefix}:echo(${caret})", display = "echo(param)",
hoverText = "echo function that returns the given string"),
            @AeFunction(syntax = "${prefix}:companyName()", display = "companyName()",
hoverText = "Returns company name") })
    public IAeFunction getFunction(String aFunctionName)
            throws AeUnresolvableException {
        if (aFunctionName.equals("echo"))
            return echoFunction;
        else if (aFunctionName.equals("companyName"))
            return companyNameFunction;
        return null;
    }
    private class EchoFunction implements IAeFunction {
        @Override
        public Object call(IAeFunctionExecutionContext aContext, List aArgs)
                throws AeFunctionCallException {
            if (aArgs.size() == 1)
                return aArgs.get(0);
            return null;
        }
    }
    private class CompanyNameFunction implements IAeFunction {
        @Override
        public Object call(IAeFunctionExecutionContext arg0, List arg1)
```

```
                    throws AeFunctionCallException {
                return "acme";
            }
        }
    }
```

Adding Global Custom Functions to the Process Server

You can add Java-based custom functions to the Process Developer embedded engine or Process Server to make them available globally to all processes and not just the contribution in which they were deployed.

First, copy your jar file to the machine that hosts the server.

Next, add custom function information to make the function known to the server. Do this within the Process Console by selecting **Admin > Global Functions**.

For help in adding details, select the Help link and view the *Global Functions* Help topic.

Writing XQuery Functions

Process Developer integrates the XQuery Development Tools (XQDT) project. Using it, you can create XQuery custom functions in your orchestration project that are automatically included in the functions list of the Expression Builder.

XQDT includes an XQuery editor with syntax highlighting, validation, content assistance, and execution support. You can easily create and test XQuery functions in the editor before using them within the Process Developer Expression Builder. Also, you can make changes to a function and save it. The updates are automatically included in your BPEL file.

Each function must be part of a library module in order for it to be included in the Expression Builder. You can create a module of functions in one XQuery file and then use the functions defined in that module in any other XQuery by using the `import module` directive.

XQDT supports XQuery 1.0 and many XQuery 3.0 constructs.

After you double-click in the functions section of the expression builder to add an XQuery function, the XQuery file is added as an import. You should pay attention to what XQuery files are referred to as imports in the BPEL process and make sure that there are no duplicates between that list and what they explicitly add as imports in the XQuery files that use the import module.

If you have an existing project (prior to version 9.0), right-mouse click on the project in the Project Explorer, and select **Add XQuery Nature**.

To get started creating an XQuery function:

1. Right-mouse click on the `xquery` folder (or another folder) in your orchestration project, and select **New > XQuery Module**.

2. Name the file with a `.xq` extension, such as `myNewFunctionModule.xq`.

3. Select one of the following:

   - Main module. Contains a query body to be evaluated

   - Library module. Contains a module declaration but no query body. A module declaration provides a URI that identifies the module for imports. Enter a prefix and namespace for the module declaration.

4. Select **Finish**, and the XQuery editor opens. Use content assist (Ctrl + spacebar) to begin a new function.

The following example shows the module declaration for a library module. This declaration is required for an XQuery function to appear in the Functions list:

```
(: The version declaration in line one is optional.
    Version 1.0 does not allow any use of version 3.0 constructs. :)
xquery version "1.0" encoding "utf-8";
module namespace xqf= "TechSupportLevel1";
declare function xqf:pingModemResult ((: $param as type, ... :)) as item() {
    insert_an_expression_here
};
```

Once you save your XQuery file, the function is automatically available in the Expression Builder, as shown.
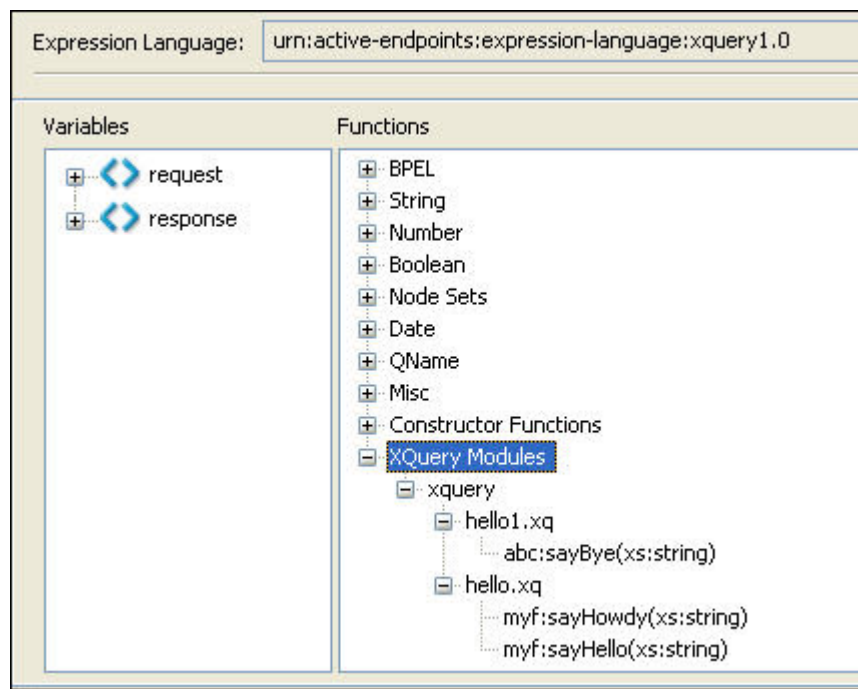
As you use a function, the corresponding XQuery module is automatically added to the process's Imports node.

Tip: In the Outline view, select the function under Imports to view its properties. If you run B-unit tests, you can add the XQuery modules as a resource and specify the Type URI as shown.

Using the XQuery Editor

The XQuery Editor opens when you create or open a file with a `.xq` extension. Note the following features:

- The editor contains a content assist feature, allowing you to select CTRL + Space to open a list of XQuery constructs, system expressions, and your own functions that exist in modules imported into the open file.

- Syntax highlighting shows various elements of XQuery expressions

- Validation occurs automatically and errors are reported

- Run As features are available for executing local scripts to test your expressions. Refer to <u>"Testing XQuery Functions in the XQuery Editor " on page 406</u>.
- You can set preferences for syntax highlighting and other features in the XQuery section of Preferences.

See also <u>"Tips on Writing XQuery Functions " on page 406</u>.

Tips on Writing XQuery Functions

Process Developer supports all XQuery 1.0 constructs and some XQuery 3.0 constructs, such as:

- Group by
- Try-Catch
- Formatting functions for date, dateTime, time, and number

In order to use XQuery 3.0 constructs, be sure to add the following declaration to your files:

```
xquery version "3.0";
```

Process Developer uses Saxon as the XQuery processor. If you are in doubt about support for a particular XQuery 3.0 construct, refer to the following at the Saxon Web site, which is at `http://www.saxonica.com/documentation9.3/changes/intro.xml`.

Testing XQuery Functions in the XQuery Editor

You can write a test script for your functions and execute the script in the XQuery Editor. Process Developer provides a default interpreter for script execution.

For example, create a new `.xq` file, and write a simple test script where you hard-code a function argument as in the *Kitty* argument below:

```
xquery version "1.0" encoding "utf-8";
import module namespace  xyz ="myFunctions" at "hello.xq";
xyz:sayHello('Kitty')
```

To run this script, right-mouse click in the editor and select **Run As > XQuery**.

Passing an External Value to a Test Script

If the test script has external variables defined, values for those variables can be passed as arguments from the Run As launch configuration. The following is a simple script that takes in an external variable, followed by a description of how to create the script argument:

```
xquery version "1.0" encoding "utf-8";
import module namespace  xyz ="myFunctions" at "hello.xq";
declare variable $ip as xs:string external;
xyz:sayHello($ip)
```

Create a Run As Configuration:

1. Right-mouse click in the editor and select **Run As > Run Configurations**.
2. Select New, and add your project name and test script name on the first page.
3. On the Arguments page, add a Script Argument, such as `ip="Kitty"`.
4. Run your named configuration.

Customizing the Interpreter Behavior

Process Developer includes a default Saxon interpreter. You can change the arguments to this interpreter by selecting **Preferences > XQuery > Interpreters**.

You can configure the default Process Developer interpreter by double-clicking it. Interpreter arguments can be changed based on the development needs. Documentation for the arguments can be found at the following URL: http://www.saxonica.com/documentation/using-xquery/commandline.xml

Do not change the first argument, which is the class name, `org.activebpel.rt.bpel.ext.expr.impl.xquery.AeQuery`.

CHAPTER 29

Custom Service Interactions

The following topics discuss custom service Interactions:

Using a REST-based Service

You can create web service interaction activities in BPEL that can handle messages based on the Representational State Transfer (REST) architecture rather than WSDL operations. This means that you do not need your own WSDL file in order to create certain types of BPEL processes. You can simply instantiate a process with a request that looks like the following example:

```
http://localhost:8080/active-bpel/services/REST/lookUpWeather/mass/waltham
```

REST is a style of architecture for describing how resources are defined and addressed on systems such as the World Wide Web. Using REST, you can send and receive REST-based messages over HTTP without an additional messaging layer such as SOAP.

You can also access partner REST services from a BPEL process.

The idea of REST is that clients and services can communicate over HTTP to exchange representations of resources. A *resource* is any item of interest, made accessible through some URL on the network. In BPEL, a *representation* is an XML document with optional attachments of a different content type.

A BPEL process can send and receive a REST request. When a REST request is sent to the process, Process Server converts it to a standard WSDL message. When the process invokes a REST service, a standard WSDL message is converted into a REST request. The response is a document representing the requested resource. The document type returned is defined by the Content-Type header of the HTTP response.

To get started with REST, you need the following:

- A BPEL project that imports `aeREST.wsdl` and `aeREST.xsd`
- For invokes you need:
  - The URL of the REST resource you will address
  - The HTTP method you want to invoke
  - The content type of the resource representation you will return
  - For XML content returned, an optional schema for the returned message (for example, atom schema, vender specific) to address the content within the BPEL process

- The query parameters specified by the REST service you want to address

See Also:

- "Creating a REST-based Receive or Invoke " on page 409.
- "BPEL REST Messages" on page 411
- "Handling of Multipart HTTP Messages" on page 411
- "Specifying Deployment Details for a REST-based Process " on page 412
- "I18N Functions " on page 249

Creating a REST-based Receive or Invoke

In a REST-based service, you can create a start activity (receive, pick, event handler) that executes the URL request that is submitted through the address field of a Web browser or an HTTP client. You can also create an invoke activity that sends a request to a web-based service via HTTP.

The receive and invoke activities must be based on a WSDL file named `aeREST.wsdl`. The port types and partner link types for this WSDL are in the System Services tree in the **Interfaces** dialog in Participants view.

In essence, the WSDL defines the `RequestResponse` operation and messages for receiving and sending REST requests. Process Developer converts an incoming REST request to a WSDL message and does the reverse for an outgoing request.

A request that targets a REST-based process looks like the following example:

```
http://localhost:8080/active-bpel/services/REST/lookUpWeather/mass/waltham
```

REST Request Message

Process Server converts the request to an input message that would have the following parts, shown in the example:

RestRequest example (Not all elements are shown. See the table below):

```
<?xml version="1.0" encoding="UTF-8"?>
  <ns2:RESTRequest xmlns:ns2="http://schemas.activebpel.org
      /REST/2007/12/01/aeREST.xsd">
    <ns2:subdomain>Search</ns2:subdomain>
    <ns2:method>GET</ns2:method>
    <ns2:pathInfo>V1</ns2:pathInfo>
    <ns2:params>
      <ns2:param name="temp" value="50"/>
      <ns2:param name="windchill" value="40"/>
    </ns2:params>
    <ns2:payload contentType="text/xml">lookUpWeather/mass/waltham?
temp=50&windchill=40</ns2:payload>
  </ns2:RESTRequest>
```

The message parts are defined as follows:

subdomain	Optional for partner service (invoke). The part that identifies one of the REST-based services that a domain can offer. For example Yahoo offers search, news, and other services. You can use the subdomain part to specify the service you want to use. In the endpoint reference for the partner link, you can add a logical subdomain value that can be replaced by the subdomain specified: `<WS-Address>nnnn.yahooapis.com</WS-Address>`.
method	The HTTP method to execute. See the `HTTP Methods` table below.

pathInfo	Optional for partner service (invoke). Parts that extends the resource details defined in WS-Address. For example, in `search.yahooapis.com`, you can specify the path info for the search service, version number, and specific type of search: `NewsSearchService/V1/newsSearch`
params	(Optional) Query parameters specified by the REST service you want to address.
headers	(Optional). HTTP headers containing connection, host, content type and other request details.
payload	(Optional) The representation of the resource. This data can be character data or XML data. If it is character data, you should include the *contentType* attribute.
queryString	(Optional) The string shown as it is formatted in the request. Contains parameters in the order they are specified.
authType	(Optional) Value populated for inbound requests (receives). This value is equivalent to the value returned by `getAuthType()` method in `javax.servlet.ServletRequest`. The value can be one of the following: BASIC, FORM, CLIENT_CERT or DIGEST
ssl	(Optional) Boolean. True if the request is sent as https.
contextPath	(Optional) The application called in the request.
requestURI	(Optional) The location of the service where the request is made.
locales	(Optional). The ISO 639 Codes for country and languages. For example, `<locale country="US" lang="en">en_US </locale>`

HTTP Methods:

HTTP defines eight methods indicating the action to be performed on the resource.

HEAD	Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.
GET	Requests a representation of the specified resource
POST	Submits data to be processed (for example, from an HTML form) to the identified resource. The data is included in the body of the request. This can result in the creation of a new resource or the updates of existing resources or both.
PUT	Uploads a representation of the specified resource
DELETE	Deletes the specific resource.
TRACE	(Not supported for receives) Echoes back the received request so that a client can see what intermediate servers are adding or changing in the request.
OPTIONS	Returns the HTTP methods the server supports. This can be used to check the functionality of a web server.
CONNECT	(CURRENTLY NOT SUPPORTED) Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

REST Response Message

The REST response message contains the parts shown below.

If you generate sample data for the response, be sure to change the `statusCode` attribute to a valid HTTP Status Code. The standard response for a successful HTTP requests is *200*. The generated integer, `"1"`, results in a fault being thrown.

```
<rest:RESTResponse
    xmlns:rest="http://schemas.activebpel.org/REST/2007/12/01/aeREST.xsd"
    statusCode="1">
<!--Optional:-->
<rest:headers>
      <!--Optional:-->
      <rest:header name="string" value="string"/>
   </rest:headers>
   <!--Optional:-->
   <rest:payload contentType="string">
         <!-- Payload information -->
   </rest:payload>
</rest:RESTResponse>
```

If `rest:payload` is not qualified, it does not inherit the `aeRest.xsd` namespace as it did in Process Developer versions prior to 9.2. If you had relied on this inheritance, your XPath expressions may no longer work until you update your process.

REST Fault Message

A sample REST fault looks like the following:

```
<rest:RESTFault xmlns:rest="http://schemas.activebpel.org/REST/2007/12/01/aeREST.xsd"
errorType="transport">
    <rest:errorCode>400</rest:errorCode>
    <rest:message>Bad Request</rest:message>
    <rest:messageDetails>The request cannot be fulfilled due to bad syntax</
rest:messageDetails>
</rest:RESTFault>
```

- The `errorType`, defined in the above sample as *transport*, can be either *transport* or *system*.

- The `errorCode` must be a HTTP Status Code, defined in the above sample as a *400* error.

BPEL REST Messages

The `aeREST.xsd` schema defines three message types:

- **RESTRequest**. This is the message received by the server from a My Role endpoint and the message delivered by the server to a REST partner endpoint.

- **RESTResponse**. This is the success response message returned to the server from a REST partner endpoint

- **RESTFault**. This is the fault returned to the server from a REST partner endpoint with the HTTP error response or other system related faults.

For examples of each message type, see .

Handling of Multipart HTTP Messages

Receives and Invokes support multipart HTTP messages.

There are two distinct classes of multipart messages, multipart/form-data and all others (for example, mixed, related, digest, parallel). Multipart/form-data represents data from an HTML form construct that can have parameters and files.

Receives

Multipart messages are parsed and populated in the `RESTRequest` message as a payload and attachments. The first textual message part is the payload all others are attachments.

For multipart/form-data, parameters are converted to `RESTRequest` message parameters. The first textual form data file becomes the payload, all others are message attachments. When there are no textual files there is no payload only attachments.

Invokes

Multipart messages are created implicitly when there are at least two of any combination of payload and attachments. The default Content-Type sent is multipart/mixed unless a multipart Content-Type header is defined in the `RESTRequest` message. An explicit Content-Type header of type multipart/form-data changes the behavior to emulate an HTML form submission. In this case all `RESTRequest` parameters in the payload are treated as HTML INPUT parameters, and attachments are treated as files.

Specifying Deployment Details for a REST-based Process

When you create a PDD file for a REST-based process, you must add the resource details and policies for the My Role and Partner Role endpoint references.

Partner Role

For a Partner role endpoint reference, select REST for the Invoke Handler. You can also add a HTTP Transport Policy Assertion to specify service timeout values.

There are no service bindings in the `aeREST.wsdl` so you can provide the endpoint reference by supplying the URL to the resource, such as:

```
 <wsa:Address>http://search.yahooapis.com:80/NewsSearchService/V1/newsSearch</
wsa:Address>
```

To use variable replacements for the specific service of the domain, you can use parameters in the address, such as:

```
 <wsa:Address>http://nnnn.yahooapis.com:80/NewsSearchService/nn/newsSearch</wsa:Address>
```

The parameters can then be specified in the request message, as described in <u>"Creating a REST-based Receive or Invoke " on page 409</u>.

My Role

For a My Role endpoint reference, ff the process is receiving REST requests, you must add a policy assertion for a REST-enabled service.

A REST service context policy example is as follows:

```
<wsp:Policy
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:aep="http://schemas.active-endpoints.com/ws/
            2005/12/policy">
  <aep:RESTenabled
        description="short description of the service"/>
  <aep:usage>
   This demo returns search results from multiple
     search engines.
   The accepted url format is:
      http://{host[:port]}/active-bpel/services/REST/
      searchDemo?query=[keyword]
   keyword  - the phrase to be searched for
  </aep:usage>
  </aep:RESTenabled>
</wsp:Policy>
```

Using an OAuth REST-Based System Service

If your business process needs a system whose resources can be accessed using OAuth authentication, you can use the OAuth system service to allow delegated access to private resources.

OAuth is an open protocol that allows secure API authorization in a simple and standard method. Users can grant third-party access to their resources without sharing their passwords and can also grant limited access, in scope or duration.

OAuth's method of delegated access to private resources uses two sets of credentials: the process consumer (in our case, a BPEL process) identifies itself using its identifier and secret, while the resource owner is identified by an access token and token secret. Each set can be thought of as a username-password pair (one for the application and one for the end-user).

Here are some examples:

- A social network can pull a user's address book from their e-mail provider to make it easy for the user to invite friends to that social network
- A photo printing service can pull a user's photos from their photo hosting service to print selected photos
- A financial aggregation service can pull a user's financial details from their banks and credit cards companies to show a combined view of that information

The OAuth system service provides authentication, access token storage for future use, and a way to do communication with the provider post authentication.

The supported OAuth service providers are listed in the PDD, as shown.
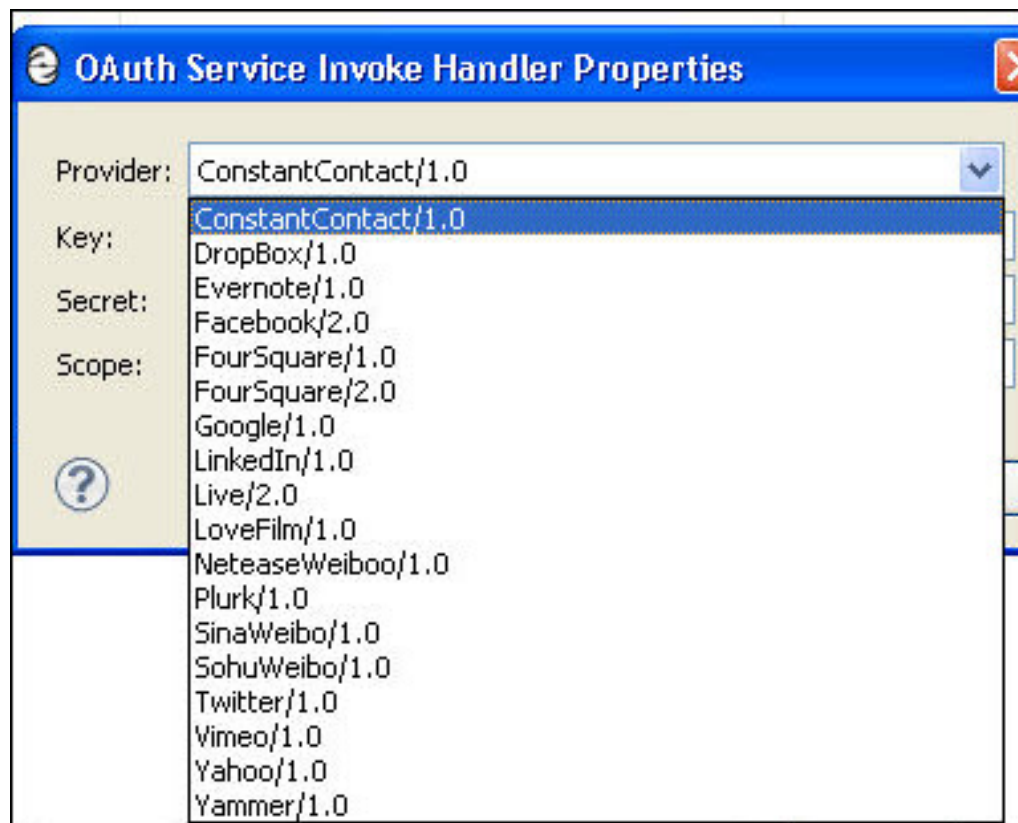
As a prerequisite, for each provider you want to access, you must register with the service provider and obtain a client identifier key and secret. These parameters are required for the Partner Role configuration in the PDD.

The OAuth system service is a client-side component of the OAuth protocol, and it abstracts OAuth tokens and request tokens (which are elements of the OAuth protocol) by a user id. Each operation takes a userId as one of the input elements to identify existing tokens. A userId can have one authorized OAuth token per OAuth provider at any given time.

Creating an OAuth Service

Use the following procedure to create an OAuth service:

1. In the Participants view of a process, create a new Partner Service Provider.

2. Select System Services from the Interfaces tree.

3. Select **OAuth**.

4. From the Partner Service Provider that you created, start with the `getAccessToken` operation to create an invoke activity. See the operation descriptions below.

The OAuth operations are as follows:

getAccessToken	Returns an OAuth access token for the given user. An access token is persisted after authorization is complete.
getAuthorizationURL	Returns the authorization URL configured for the OAuth provider. Process Developer maintains this URL for each OAuth provider configuration listed in the PDD. This authorizationURL is invoked by the user (outside of Process Developer), and during this process, the OAuth provider lets the user know that Process Developer is trying to access the account and gets the user's content.

Once the user approves the access, the provider returns a verifier string/code to the user. In the case of OAuth2.0, some providers require a redirect URL to which the request is redirected along with verifier string as a request parameter. For providers (usually OAuth2.0 providers) who require a redirect URL, it can be specified in the PDD along with other credentials. |
| authorize | Takes the verifier string (given by the OAuth provider as a response to the authorizationURL) and authorizes with the OAuth provider. It also stores the OAuth access token returned by provider. This token is used further to access the protected resources on the provider.

Once this token is generated, OAuth initial setup workflow is over. This is typically a one-time effort. However, it has to be done once the OAuth access token is expired (its lifetime is decided by OAuth provider). |
| sendRequest | If the OAuth token for the given OAuth provider and user is present in Process Developer, this operation sends a request to access protected resources on OAuth provider system. This should be preceded by an *authorize* call, which generates and stores token in Process Developer. |

The following illustration shows a sample implementation for an OAuth service.
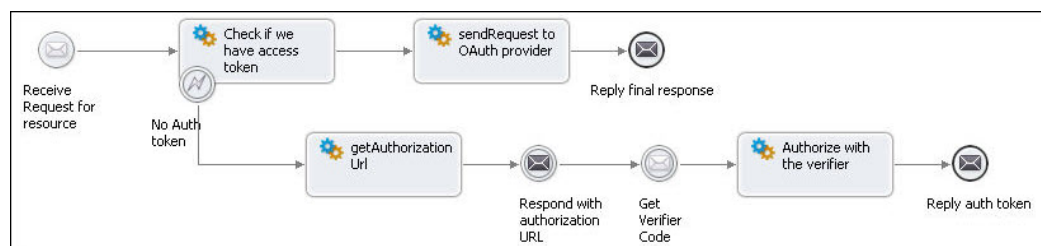
For deployment details, see .

Specifying Deployment Details for an OAuth Service Provider

This topic discusses OAuth Invoke Handler Input Parameters.

For an overview, see .

When you create a PDD file for an OAuth provider, you must add the appropriate parameters for the Partner Role endpoint reference.

For a Partner role endpoint reference, select Oauth Service for the Invoke Handler.

Fill in the following configuration parameters for the service:

OAuth Provider	Select a provider from the list
Key (required)	Represents the OAuth client ID, used to authorize a client. You must follow the procedure from the provider to obtain the key. For example, LinkedIn requires you to fill in a form online to receive a key and secret.
Secret (required)	Secret text given by the OAuth provider. Used during authorization.
Scope (optional)	Scope represent the subdomain of the OAuth provider that the OAuth client wants to access. Ex: `https://docs.google.com/feeds/` is a scope value to access the Google docs feed. This is optional from the OAuth system service perspective, but can be mandatory based on the target OAuth provider.
RedirectURL (optional)	This is usually required by OAuth2.0 providers. The response of authorization with a verifier string is redirected to this URL. Usually the verifier string is a query parameter (in the form `url?code=verifierString`).

Adding a Custom Provider

It is possible to integrate with a custom OAuth provider (one that is not provided in the default list) and add it to the list in the PDD. To add a custom provider, create a configuration file (`.oap` file) in your project's deploy folder describing one or more endpoints for providers.

For example, to add the *ACMEOAuthProvider*:

```
<?xml version='1.0' encoding='UTF-8'?>
<oaconfig:oauthProviders xmlns:oaconfig="http://schemas.activebpel.org/OAuth/2011/08/01/
aeoauthProviderConfig.xsd"
targetNamespace="http://schemas.activebpel.org/OAuth/2011/08/01/
aeoauthProviderConfig.xsd">
  <!-- name of the provider and version. Any text and oauth
       version -->
  <oaconfig:provider name="ACMEOAuthProvider"
            oauthVersion="1.0">
    <oaconfig:requestTokenEndpoint
        verb="POST"> https://oauth.acme.com/ws/oauth/request_token
</oaconfig:requestTokenEndpoint>
    <oaconfig:accessTokenEndpoint verb="POST">
            https://oauth.acme.com/ws/oauth/access_token
    </oaconfig:accessTokenEndpoint>
    <oaconfig:authUrl verb="GET">
          https://oauth.acme.com/ws/oauth/confirm_access?oauth_token=%s
    </oaconfig:authUrl>
  </oaconfig:provider>
</oaconfig:oauthProviders>
```

Note that the placeholders (`%s`) should be given in the URL templates so that the OAuth token is substituted by the OAuth system service.

Once the configuration file is part of project, it is detected, and the PDD's OAuth provider list is updated with the local configuration. Note that the extension of the file must be `.oap`. The file can hold any number of custom OAuth provider configurations.

A configuration file for out-of-the-box OAuth providers can be found in the Process Server catalog with the location hint `project:/com.activee.rt.oauth.services/config/ae-oauth-providers.oap`.

Using a Java Messaging Service Invoke Handler

Process Developer supports a Java Messaging Service (JMS) invoke handler for invoking endpoints. This means that you can bypass the standard Process Developer engine invocation framework of SOAP over HTTP with a SOAP over JMS invocation or a plain XML message over JMS. With a JMS invocation, your process can communicate with any of the Message-Oriented Middleware (MOM) applications that your server supports. This means when you invoke a partner service you can:

- Connect to MOM queues
- Set the names of MOM queues on which the server should return the response
- Create the MOM message that contains the SOAP envelope
- Drop the MOM message that contains the SOAP envelope

Providing Deployment Description for JMS Invoke Handler

In the Process Deployment Descriptor file of a BPEL process, you can specify the details of a JMS invoke handler that provides Metro with the details needed to call the JMS transport mechanism.

On the Partner Links page of the PDD editor, select jms as the Invoke Handler for the partner role.

In the Endpoint Reference text box, add the details specific to JMS transport.

The details you can provide include:

- Queue or topic look up name
- For a request/response service, the reply queue as a `wsa:ReplyTo header` as a `ReferenceParameter` for routing responses (see example below)
- Additional JMS message properties, included as `wsa:ReferenceParameters` on the partner endpoint
- Optionally, you can include a JMS policy assertion on the partner role (See examples below)

Providing the Endpoint Reference Details

In Process Server, SOAP messages delivered over JMS use WS-Addressing headers to route requests to the appropriate service.

The target queue and, optionally, the service name of a JMS invoke is specified in the Address element of the endpoint reference, formatted as `<queue JNDI name>?<service name>`.

Example 1: Looking up a queue using a JNDI Name

```
<wsa:EndpointReference>
  <wsa:Address>queue/com.activee.jms.bpel.queue
      ?WSSReceiverService</wsa:Address>
</wsa:EndpointReference>
```

This translates into looking up the queue using a JNDI name of:

```
queue/com.activee.jms.bpel.queue
```

with a target service name of `WSSReceiverService`. The service name is used by the message receiver on the other side to determine the target service you are invoking.

Example 2: Using a <wsa:ReplyTo> for Two-Way Operation

For synchronous operations, where a durable response is not required, the JMS mechanism of a temporary queue handles messages for an invoked service. Note that the temporary queue cannot be accessed during recovery or on another node due to failover.

If the activity is long running, or requires failover, you can specify the response location in a `<wsa:ReplyTo>` element in the Endpoint Reference definition. For example:

```
<wsa:EndpointReference>
  <wsa:Address>...</wsa:Address>
   <wsa:ReferenceProperties>
    <wsa:ReplyTo>
     <wsa:Address>queue/com.activee.jms.reply.queue/
          </wsa:Address>
    </wsa:ReplyTo>
   </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

Notice that the reply address specifies only the queue name with no service. The service name is not necessary for a two-way response message.

Example 3: Using a <wsa:ReplyTo> for CallBack Operation

If the operation is a one-way with a callback, specify the `<wsa:ReplyTo>` endpoint as follows:

```
<wsa:ReplyTo>
 <wsa:Address>queue/com.activee.jms.reply.queue
    ?JMSCallbackService
      </wsa:Address>
</wsa:ReplyTo>
```

Example 4: Adding JMS Properties to the Endpoint Reference

If you need to populate additional JMS string properties on an invoke, they should be included as `wsa:ReferenceParameters` on the partner endpoint. These properties are also included as SOAP headers if sending as a SOAP envelope.

```
<wsa:ReferenceParameters>
 <wsa:ReplyTo></wsa:Address>...</wsa:Address>
    </wsa:ReplyTo>
  <abx:param name="someProperty" value="value"
        xmlns:abx="http://www.activebpel.org/bpel/extension" />
</wsa:ReferenceParameters>
```

Adding a JMS Delivery Options Policy Assertion to a Partner Role

The message sender (partner role) controls the messaging options used for JMS. When Process Server receives a two-way request, the reply is sent mirroring the options used for the request. For example, if the request message is a bytes message formatted as plain XML, the response will likewise be a bytes message formatted as plain XML. The options used for sending a JMS request are controlled through a WS-Policy assertion on the endpoint. You can specify additional attributes for JMS message delivery as a partner role policy assertion.

The basic policy attributes are displayed in the JMS Delivery Options Policy Assertion dialog. For instructions on using this policy assertion, see "Adding Policy Assertions " on page 357.

Message Type: Text (default) or Bytes	A text message content is XML text. A bytes message content is a byte array.
Message Format: SOAP (default) or XML	SOAP the default format. SOAP refers to a SOAP envelope as message payload. For SOAP messages, all WS-Security features are available to provide message level authentication, encryption and signature support. XML is either a request or response document as plain XML with no SOAP envelope. For plain XML messages, message level security is not available and you must control access by configuring authorization restrictions on JNDI lookups for the destinations through your application server's administration console.
Priority (integer)	If desired, specify a non-negative integer for a message handling priority. The default value is 0, which means that the default for the provider is used. JMS defines a 10-level priority value with 0 as the lowest and 9 as the highest. Clients should consider 0-4 as gradients of normal priority and 5-9 as gradients of expedited priority. Priority is set to 4, by default.
Persistent Delivery Mode	This setting controls whether or not the JMS provider persists messages to storage for all processes. The default is *persistent*. Select *persistent* to instruct the JMS provider to ensure that a message is not lost in transit in case of a JMS provider failure. A message sent with this delivery mode is logged to stable storage when it is sent. Note that persistent delivery requires that your JMS provider be configured with storage. Also, there is usually a performance hit with persisting messages. Select *non-persistent* if desired. This mode does not require any knowledge of how a provider is configured for storage.
Time to Live (ms)	This setting allows you to specify the amount of time an unconsumed message remains on a queue (in milliseconds). If a message will become obsolete after a certain period, you may want to set an expiration time. The destruction of obsolete messages conserves storage and computing resources. The default setting is 0, which means that the default for the provider is used. Typically, this means that messages never expire and remain on the queue forever.
JMS Manager ID	In the Process Console, you can add multiple JMS Provider configurations. If you want to use one other than the default, you can specify the Manager name for a JMS Provider that has been configured.

Other attributes, such as the correlation id and expiration, can be set dynamically at runtime through a copy operation to the partner endpoint.

A `jmsCorrelationID` correlates response messages for two-way operations. By default, Process Server use the `wsa:MessageID` of the request as the unique correlation Id. You can use the default correlation id with an external consumer, provided that consumer will propagate the `JMSCorrelationID` from the request. If you notice a problem with invoke timeouts, a missing/incorrect correlation id can be the cause.

If the consumer will not propagate the correlation ID from the request, you can set an explicit `jmsCorrelationID` as an override in this policy. Note that using a static value becomes problematic if there could be more than one invoke waiting for a response on the same queue. If responses all use the same correlation id, there is no way to distinguish which response goes where. A workaround for this is to use a temporary queue for receiving responses. If using a temporary queue for responses, a correlation id is not necessary since there will be a unique temporary queue for each request-response pair.

Expiration is time in milliseconds from its dispatch time that a produced message should be retained by the message system.

Here is an example of a policy assertion added to a partner endpoint dynamically in a copy operation:

```
<partnerRole endpointReference=
   "static" invokeHandler="jms:Address">
   <wsa:EndpointReference>
   <wsa:Address>queue
       /com.activee.jms.bpel.queue?JMSOneWayService
   </wsa:Address>
   <wsp:Policy xmlns:abp="http://schemas.active-endpoints.com
      /ws/2005/12/policy"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <abp:JMSDeliveryOptions
      jmsManagerID="JMSMessagingManagerName"
      jmsMessageType="bytes"
      jmsMessageFormat="xml"
      jmsPriority="1"
      jmsCorrelationID="12345678"
      jmsExpiration = "9999999999"/>
   </wsp:Policy>
   </wsa:EndpointReference>
</partnerRole>
```

Configuring the Messaging Service in the Process Console

To take advantage of JMS messaging, your Process Server administrator must enable the Messaging Service within the Process Console. See the *Process Console Help* for details.

CHAPTER 30

Process Exception Management

Process exception management is a feature available on Process Server. Benefits of this feature include suspending processes when uncaught faults occur as well as providing alerts of their occurrences. The Process Developer helps you use this feature in several ways.

The following topics discuss process exception management:

- "What is Process Exception Management " on page 420
- "Suspending a Process Programmatically with a Suspend Activity" on page 421
- "Alert Service " on page 468

What is Process Exception Management

Process exception management involves managing exceptions that occur in running processes. The exceptions typically manifest themselves as faults that are not caught by normal fault handlers; that is, they were not planned for. They are the result of unexpected errors in process execution, such as bad data or problems with external systems.

Process exception management allows the process to be suspended at the faulting activity rather than be terminated abnormally. In addition to the automatic suspend on uncaught fault, you can check for exception cases programmatically in BPEL and use the Suspend activity to suspend the process. In either case a user, who could be alerted of the problem, can then correct the errors in place and resume the process.

You can perform process exception management in the Process Console. For details, see "Suspending a Process on Uncaught Faults " on page 420.

For details, see:

- "Suspending a Process on Uncaught Faults " on page 420
-
- "Alert Service " on page 468

Suspending a Process on Uncaught Faults

Suspending a process on an uncaught fault applies to processes running on Process Server.

Use the following techniques to suspend all or individually selected faulting processes on Process Server:

- "Making Individual Processes Eligible for Suspension on Uncaught Faults " on page 421
- "Making Individual Processes Eligible for Suspension on Uncaught Faults " on page 421
- "Suspending a Process Programmatically with a Suspend Activity" on page 421

The `Suspend on Uncaught Fault` setting requires either Full or Persist, process persistence. For information on persistence, see the Process Persistence section within the "General Deployment Options " on page 348 topic.

Making all Processes Eligible for Suspension on Uncaught Faults

Suspending a process on an uncaught fault applies to processes running on Process Server.

According to the WS-BPEL 2.0 specification, a process with an uncaught fault terminates with a termination handler. However, Process Server allows processes to be suspended when an activity is faulting if the fault does not have a corresponding catch (that is, uncaught). Before completing execution of the activity, the server stops the process so that you can retry or complete the activity.

You can enable this setting in the Process Console. For details, see the *Process Console Help*.

If desired, you can override the server setting for individual processes. For details, see "Making Individual Processes Eligible for Suspension on Uncaught Faults " on page 421.

Making Individual Processes Eligible for Suspension on Uncaught Faults

Suspending a process on an uncaught fault applies to processes running on Process Server.

On the server, there is a configuration setting to make all processes eligible for suspension on an uncaught fault. However, you can override the server setting in the process deployment descriptor (PDD) file of an individual process. For details, see "General Deployment Options " on page 348.

See also "Making all Processes Eligible for Suspension on Uncaught Faults " on page 421.

Suspending a Process Programmatically with a Suspend Activity

For details, see *Suspend*.

CHAPTER 31

Creating Reports for Process Server and Central

Add resources to deploy to the server.

Many of the pages in the Process Console display information about deployed and active processes, as well as active tasks from BPEL for People processes. Many of these pages contains reports designed in Process Developer using the Business Intelligence and Reporting tools (BIRT) in Eclipse.

The reports are designed essentially by making queries to the database and returning the results in the layout desired, such as tables and charts.

You can create your own reports in Process Developer and then deploy them to the server and to Process Central. As processes execute on the server, the database is updated and so is your report.

After you design and save a report, you can deploy it to the Process Server in a BPR file. A report is a resource that is stored on the server. The reports you create appear on the Report page of Process Console, along with the standard reports.

If you want to also display reports in Process Central, you must create an Process Central Configuration file for it. For details, see "Creating a Process Central Configuration File " on page 449.

The report definitions are stored in the Process Console catalog, so you can view, update, and delete the XML source code for each report, if desired.

A good way to get started designing reports is to look at the sample reports included with Process Developer.

Topics discussing reports are as follows:

- "Creating the User Reports Orchestration Project " on page 423
- "Using the Process Developer Report Template " on page 423
- "Using the Process Developer Data Source " on page 425
- "Creating a Data Set from the Process Developer Data Source " on page 428
- "Understanding the Process Developer Data Model " on page 429
- "Deploying a Process Developer Report" on page 429
- "Updating or Deleting a Deployed Report " on page 430
- "Reporting Service " on page 430

Creating the User Reports Orchestration Project

To view and deploy example reports, you can create an orchestration project from a template:

1. Select **File > New > Orchestration Project**.

2. Name the project *Sample Reports* or *User Reports* or similar, and click Next.

3. On the New Orchestration Project From Template page, select **User Reports**.

In the orchestration project, you will see a folder called `report`. Double-click on one of the reports in the folder to open the Report Design perspective.

The sample reports are based on the Process Developer Template. For details, see "Using the Process Developer Report Template " on page 423.

If you want to deploy the sample reports to the Process Developer embedded server, see "Deploying a Process Developer Report" on page 429.

Using the Process Developer Report Template

You can create a new Process Developer report using the Process Developer Template.

1. Select **File > New > Other > Business Intelligence and Reporting Tools > Report**.

2. Name a new report, such as *new_report.rptdesign*, and click Next.

3. From the New Report page, select the Process Developer Template, as shown in the illustration.
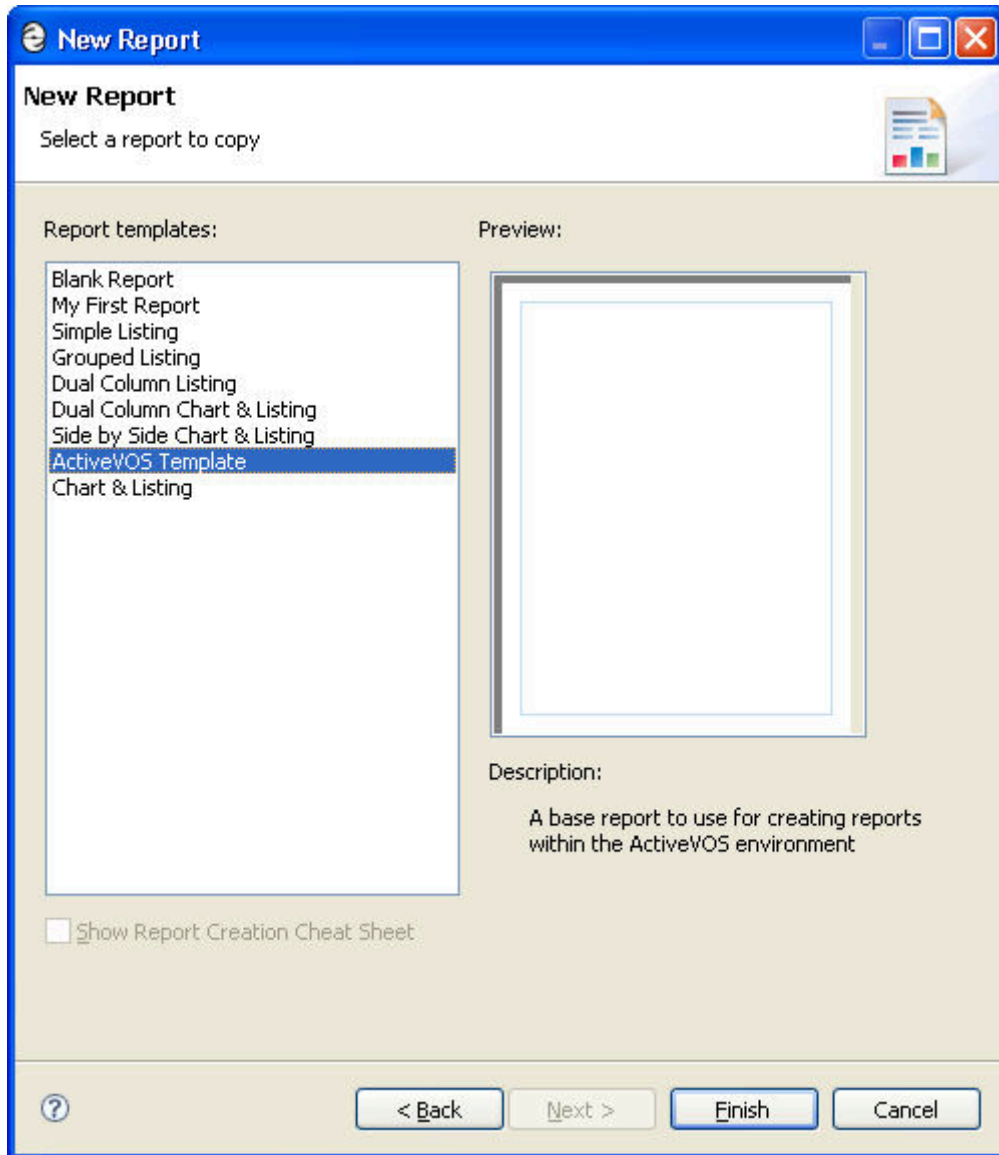
4. Select **Finish** and accept the suggestion to open the report design perspective.

You can view the template components in the Outline view, as the illustration shows:
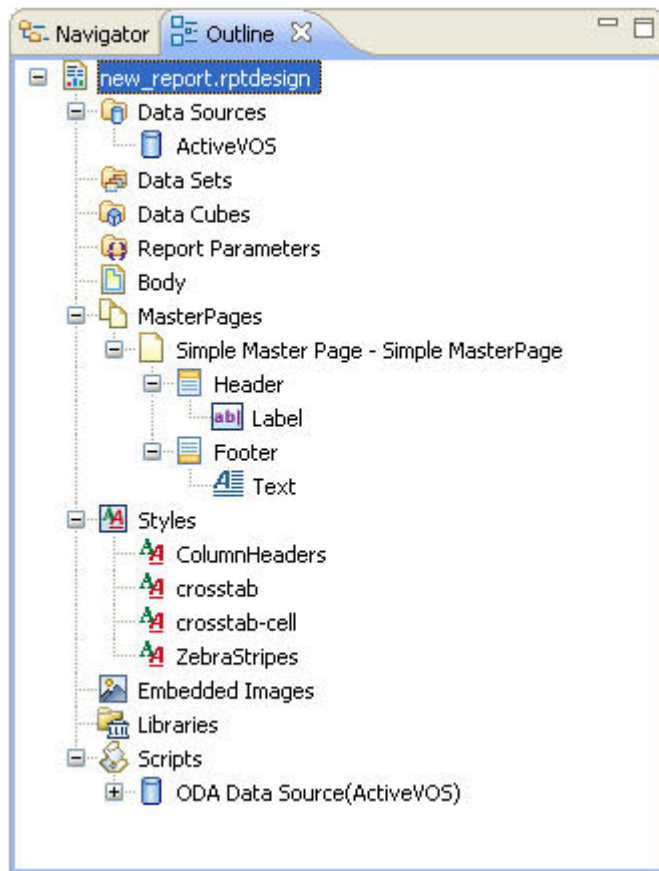
The template contains the following:

- **Data Source Connection**. This connection is pre-defined to point to the Derby database included with the Process Server. Derby includes an embedded JDBC driver. For details on the Data Source, see "Using the Process Developer Data Source " on page 425.

- **Master Page Header and Footer**. Double-click on a header or footer to view or change its properties.

- **Styles**. Several table row styles are included for reports with a data column layout.

- **Script**. When reports are deployed and running on the Process Server, the script, which is embedded in the report XML file, detects the Data Source connection information by using the supplied `AeBirtContext` class. For details, see "Deploying a Process Developer Report" on page 429.

Using the Process Developer Data Source

Included in the Process Developer Template is a connection to the Data Source that is pre-defined to point to the Derby database included with the Process Server. Derby includes an embedded JDBC driver.

Note that Derby accepts only one connection at a time, so you cannot run the Process Developer embedded server while you are designing reports.

To view the connection configuration, open the Outline view, and expand Datasources. Double-click on the Process Developer data source, as shown in the illustration.
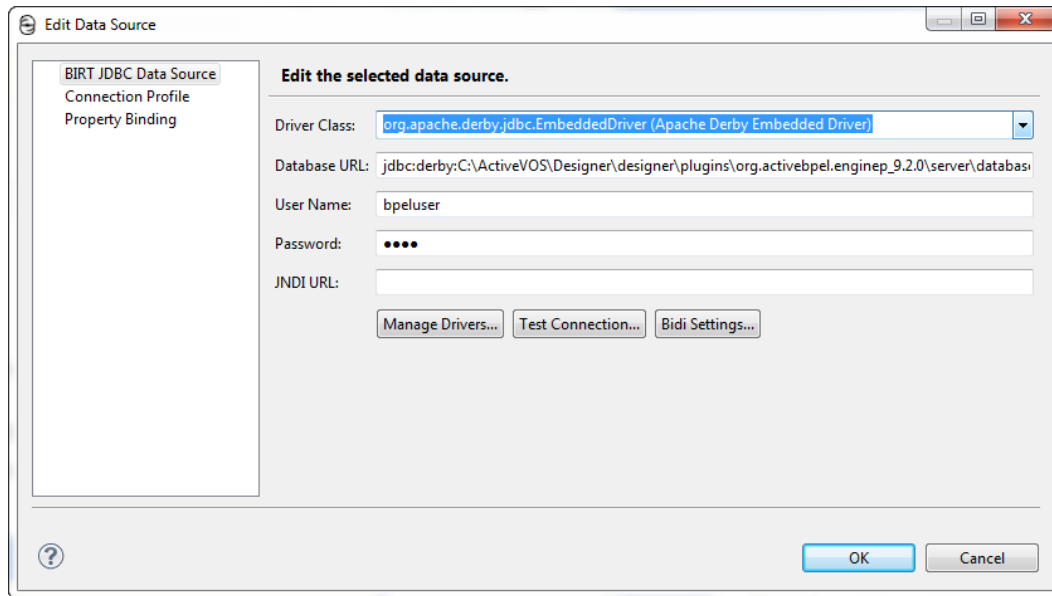
To change the configuration for a different Process Server, you must add the same Driver Class and Database URL that were configured for that server.

For example, if you have purchased Process Server with Apache Tomcat and want to configure the MySQL database on Apache Tomcat server, you would use the following settings:

Driver Class:

```
com.mysql.jdbc.Driver
```

Database URL:

```
jdbc:mysql://localhost:3306/ActiveVOS?useUnicode=true
    &characterEncoding=UTF-8&characterSetResults=utf8
```

If your server database connection was configured with any changes to the DDL script, or any configuration details, such as the User Name and Password, you must also update these settings. The User Name and Password shown are the default for all servers.

The JNDI URL for the data source is detected at runtime.

Use the three buttons as follows:

- **Manage Drivers**: Pressing this button displays a dialog with two tabs. The first lets you manage JAR files used by your data source.
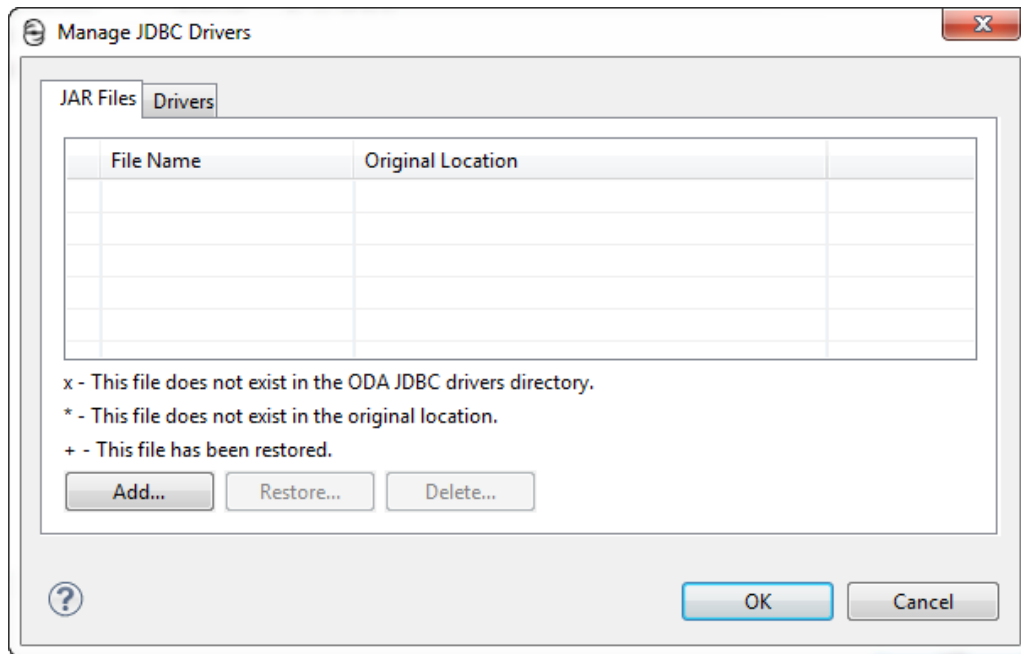
The second lets you manage its drivers:
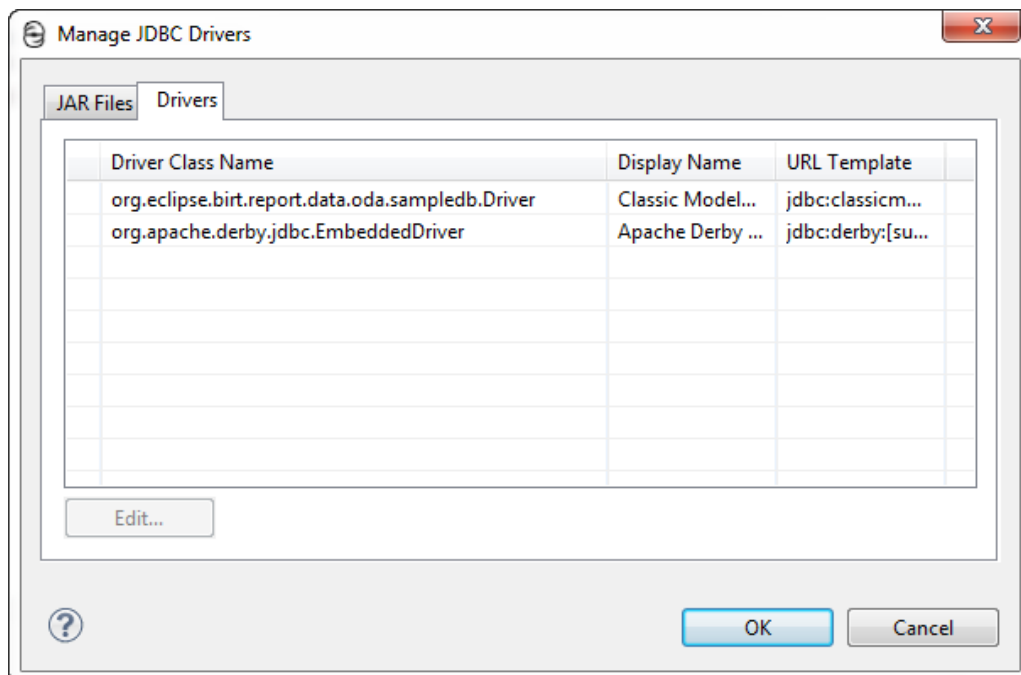
- **Test Connection**: Pressing this button tests the connection to the data source.

- **Bidi Settings**: (bidi stands for "bi-directinal") Pressing this buttons displays the Advanced Bidi Data Source Settings dialog box. Use the controls within it to set metadata and content Bidi formats.
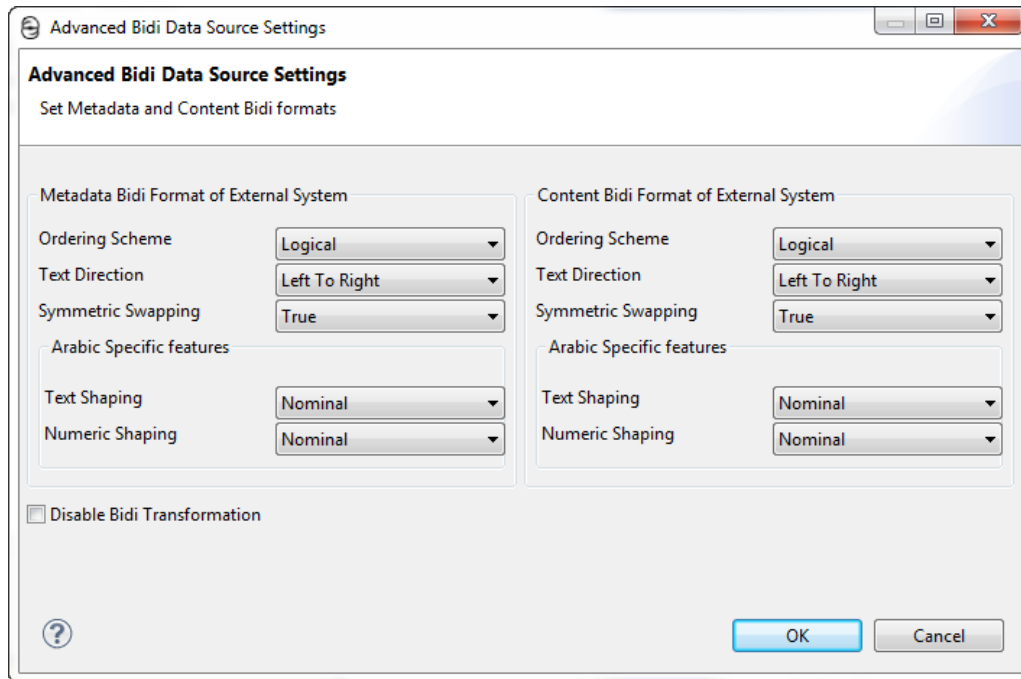
See also "Creating a Data Set from the Process Developer Data Source " on page 428 and "Understanding the Process Developer Data Model " on page 429.

Creating a Data Set from the Process Developer Data Source

To build a Process Developer report, you create a data set. A data set specifies what data to retrieve from the data source.

You can view a sample data set from a report in the User Reports Orchestration Project. To get started, see "Creating the User Reports Orchestration Project " on page 423.

Open a report from the report folder of the User Reports Orchestration Project. Then double-click on the data set to view its properties, as shown in the following example.
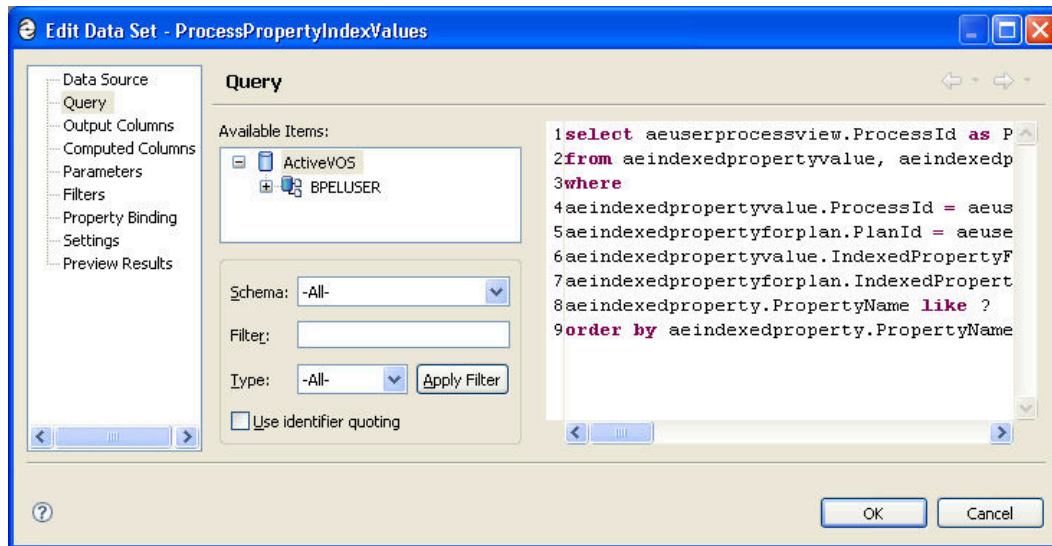
For details on using data, see the *How to create a data set* help topic in the *BIRT Report Developer Guide*.

To figure out what data you want to retrieve, see <u>"Understanding the Process Developer Data Model " on page 429</u>.

Understanding the Process Developer Data Model

Reports you create for the Process Server can retrieve data from the Process Developer Data Source that is included in the Process Developer report template. To create a report design based on this template, see <u>"Using the Process Developer Report Template " on page 423</u>.

To create a data set based on the data source, you need details about the Process Data Model and Human Task Data Model.

To view discussions as well as a sample entity relationship diagrams, see Process Developer Data Source Models elsewhere in this help.

To view and use the complete Data Source model associated with your version of Process Developer, visit the Developers SDK page on `http://www.activevos.com`. Scroll to the Reporting section.

Deploying a Process Developer Report

A report is a resource you can deploy to the Process Server and also display it in Process Central, if desired. It is deployed the same way that a BPEL/PDD or other resource is deployed by using the Contribution - Business Process Archive export.

If you are not deploying to the Process Developer embedded server that contains the Derby database, be sure to modify the Data Source Connection details before deployment. For more information, see <u>"Using the Process Developer Data Source " on page 425</u>.

If you want to display the report in Process Central, you must also create a configuration file, described in <u>"Creating a Process Central Configuration File " on page 449</u>

During deployment, you can add a Group Name and Description for each report so that your reports are grouped together on the Process Server Report page.

For instructions on deployment, see "Deploying Additional Resources " on page 376.

A report is deployed as an XML file that you can view in the XML Source tab of the Report Process Developer. It is possible to edit the XML file on the server, as described in "Updating or Deleting a Deployed Report " on page 430.

Updating or Deleting a Deployed Report

After you deploy a report to Process Server, you can view the XML source of the report from the Process Console Catalog page.

To update or delete a deployed report definition:

1. Start up the Process Server and open the Console.

2. In the Console, go to **Catalog > Resources > Report Definitions**.

3. Select a report and edit it, if desired.

4. Select **Update** to submit your changes or select **Delete**, if desired.

Reporting Service

Process Server supports a reporting service. This means that you can create a BPEL process that sends a request to the server to take a snapshot of a deployed report, save the report as PDF, MS Word, or other format, and return it to the process. You use an invoke activity for this purpose. Within your BPEL process, you provide programming logic to email the report, use it in a People activity, or send it to an endpoint based on an invoked service. You can schedule the process to run daily (or other frequency) to have a daily report emailed.

Any BPEL process that implements report-based activities imports the WSDL provided with *Informatica Business Manager*. The name of the WSDL is `reporting.wsdl`, and its port type, `reportingPT`, is in the **Participants** view in the **New Partner Service Provider** wizard under **System Services**. The WSDL contains a *report* operation.

```
<invoke
    inputVariable="reportMessageInput"
    operation="report"
    outputVariable="reportMessageOutput"
    partnerLink="Provider" />
```

To create a BPEL process that requests a report:

1. In the Project Explorer, create a new Orchestration project.

2. Select **File > New > BPEL Process**.

3. Name the process and click **Finish**.

4. In the Participants view, select New Partner Service Provider.

5. In the wizard, select System Services.

6. Select *Execute Report*, and select **Finish**.

7. From Participants view, drag the **report** operation to the canvas to create an invoke activity.

8. Fill in the Input values as described below.

9. Add the additional activities and programming logic to copy the report to an email, People activity, or other service.
   Tip: Use the attachment function, `setAttachmentProperty`, to give the report a meaningful name when it is attached to an email.

10. Add fault handling by using the WSDL fault message fault name `reportFault` and variable as desired.

On the Process Console, go to the URN Mappings page and update the URL for Internal Reports, if needed. The default is localhost:8080.

Input Message for Reporting Service

The input message for the invoke activity has the following parts, as shown in the example.

```
<ns2:reportMessageInput xmlns:ns2=
    "http://schemas.active-endpoints.com/reporting/2009/05/reporting.xsd">
    <ns2:report>ServiceActivityDetail.rptdesign</ns2:report>
    <ns2:format>pdf</ns2:format>
    <ns2:parameters>
      <ns2:param name="Service_Name" value="OrderProcessSvc"/>
    </ns2:parameters>
    <ns2:otherOptions>
      <ns2:option name="BIRT Parameter" value="true"/>
    </ns2:otherOptions>
</ns2:reportMessageInput>
```

Specify the input parameters for the invoke activity as follows.

\<report>	(Required). Name of the report design template. This is a report deployed to Process Server. The report names are visible in the Process Console Catalog
\<format>	(Required). Format of the generated report. Valid formats are: - pdf (default) - doc (Word) - xls (Excel) - ppt (Powerpoint) - ps (postscript)
\<parameters>	Each report can have its own set of parameters that are input to the report. For example, specify the name of a deployed service to get a report about one particular running process. Parameters for the standard reports are described below.
\<otherOptions>	Options for producing the report. There are many options, and they are documented in the BIRT documentation. For details, here is a link. `http://www.birt-exchange.org/devshare/ deploying-birt-reports/492-birt-viewer-2-3-user-reference/#description`

Reports and Their Parameters

You can use the reporting service to access your own custom reports or the Informatica Business Process Manager standard reports that are displayed in the Process Console.

In the Process Console, you can view a list of Informatica Business Process Developer report designs. You can then select a report design to name in the `<report></report>` element of the `reportMessageInput` variable.

To view a list of report design names, do the following:

1. Ensure that the server is running.

2. Open the Process Console.

3. Select **Catalog > Resources > Report Definitions**.

4. In the Filters table, disable the check box for Hide System and select Submit.

Some of the reports have parameters, and you can use them in the `<ael:param name="string" value="string"/>` element of the `reportMessageInput` variable.

The following reports contain parameters and valid values.

Active Tasks

- RP_State - (To see a complete list of states, go to the Active Tasks page of Process Console and select the Run Report button. In the RP_State picklist, you can view the 14 different states.)
- RP_CreatedAfter - Date
- RP_CreatedBefore - Date
- RP_CompletedAfter - Date
- RP_CompletedBefore - Date
- RP_Name - String
- RP_Owner - String
- RP_SearchBy - String
- RP_OwnerType - integer
- RP_IdentityType - integer
- RP_NamesText - String

Server Log

- RP_LogLevel VERBOSE = 1, INFO = 5, WARNING = 6, ERROR = 7 CRITICAL = 8
- RP_LogSource USER = 8, MAINTENANCE = 9 and SYSTEM = 10
- RP_LogService ENGINE = 3, ALERT = 4, MESSAGING = 5, TASK = 6, VERBOSE = 7, EVENTING = 8, EMAIL = 9, IDENTITY = 10
- RP_LogEngineId integer greater than or equal to 1

See also <u>"Email Service " on page 479</u>

CHAPTER 32

Business Event Processing

Business event processing allows you to monitor activity of running or completed BPEL processes. For example, you might want to know, "What is the average time for service X to process an order?" where service X is an invoke activity in your process, and you are evaluating a given number of completed processes over a certain time.

Another example of a business event to process is, "Alert me if the average time it takes to claim a task exceeds N minutes." In this case, you can send an alert when a certain business condition is met.

The basic steps required for business event processing are the following:

- Define an *event*. Refer to "Defining an Event in the Process Deployment Descriptor " on page 433.
- Deploy the event-based processes
- Create an event action process that runs based on a matching business condition
- Design and deploy the event-action process to manage and monitor events
- Deploy and run the processes that have defined events

Some of the benefits of using eventing include:

- Monitor processes in action and spot abnormalities
- View reports of activity
- Perform the analysis of a process that has already completed in order to view the cause of a problem

The following topics discuss business event processing:

Defining an Event in the Process Deployment Descriptor

Defining an Event in the Process Deployment Descriptor

If desired, you can define a process-specific event to be used in event processing. This type of event is based on a specific process activity or link. An event definition includes a BPEL object plus one or more life cycle states, such as *executed* or *faulted*, and can optionally include data from an indexed property that is available when the event occurs.

You define a process event in the Eventing tab of the Process Deployment Descriptor Editor.

Defining a process-related event is optional. If desired, you can use system-defined events. For details, see "Using System-Defined Events " on page 434.

To define a process-related event:

1. Create a new Process Deployment Descriptor, described in "Creating a Process Deployment Descriptor File " on page 347.

2. If desired, define one or more indexed properties. An indexed property of a process variable can be used in the definition of an event. For details, see "Adding Indexed Properties " on page 370.

3. Select the Eventing tab in the PDD Editor.

4. Select **Add**.

5. In the Event Definition dialog, type in a Service Name for an event. This is the name of the event-action service that responds to this event. See "Creating an Event-Action BPEL Process " on page 436.

6. In the Location picklist, select a BPEL object, such as an activity or link. For example select an activity:

   For convenience, select the Dialog button to open the Find Business Event Source dialog. This dialog displays named activities and links; that is, activities and links for which you replaced the generic label (such as `receive` or `l2`) with a meaningful name. Use wildcards to locate an activity or link name.

7. Select one or more activity states to trigger the event. For a description of states, see "Activity States Event Properties Task States and Task Event Types " on page 437. For example, select *executed*.

8. If desired, add a business property mapped to an existing index property. A business property allows you to return the data value that exists in a variable at the activity state selected for the event. For example, you can find all faulting processes where the *customerId* is *101*. To add a business property:

   a. In the Business Properties panel, select **Add**.

   b. In the Indexed Properties column, select the pick arrow to select an indexed property created on the Indexed Properties tab of the PDD.

   c. In the Business Alias column, select Property Name and notice that the indexed property name appears.

Using System-Defined Events

Using System-Defined Events You can select a built-in system event to be used in event processing. This type of event is based on well-known Process Server events.

Defining a system event is different from defining an event in the PDD of a process. You do not use the Eventing tab of the PDD for system events. Instead, you create a process based on the Event Action system service, and in the Partner Links tab of the PDD, you specify a required service name for the My Role partnerlink. The details are provided below.

The following system events are available

- AeEngineEvent
  See the Engine Event table below for a description of this event. In order to receive the `AeEngine` event, you must deploy a process, based on the Event-Action system service. The PDD of this process must contain the required service name `AeEngineEvent` (with `portType EventActionPT`). For details, see "Creating an Event-Action BPEL Process " on page 436.

  When using this event, be careful using the `getProcessState` admin API call if another node could suspend the event. For example, if a node tries to load a process from a second node within the default 10 second process idle timeout, the engine will suspend threads in the node running the process so that the requesting node (node 1) can load them. Being suspended, however, triggers an engine event process that tries to get the process state. However, trying to get the process state from the node 1 also triggers a suspend, which again triggers the engine event process. An attempt is now made to get the process state from node 1. These actions repeatedly suspend the parent process over and over, which means the process will not return to a running state.

- Task-Based Engine events
  See the Task-Based Engine Events table below for descriptions. In order to receive a task-based event, you must deploy a process, based on the Event-Action system service, containing one of the following required service names:

  - *AeHumanTaskCreated*. Use this service name only for the AeHumanTaskCreated event

  - *AeHumanTaskEvent*. Use this name only for the AeHumanTaskEvent event

Engine Event

Event Name	Description	Event Properties
AeEngineEvent	Fired when an engine event is fired. Engine events consist of the following. To filter an engine event, refer to its integer value: - PROCESS_CREATED (0) - PROCESS_COMPLETED (1) - PROCESS_SUSPENDED (2) - PROCESS_RESUMED (3) - PROCESS_STARTED (4) - PROCESS_RECREATED (5) - PROCESS_UNDER_COORD (6) - PROCESS_FAULTED (7) - ENGINE_STARTED (100) - ENGINE_STOPPED (101) - ENGINE_FAILED (102)	processId (long) state (int). The integer values are shown to the left. processName (QName) (Cannot be used in a "compare" statement) processLocalName(string) processTargetNamespace timeStamp (Date) invokerLocationPath (String) (ProcessUnderCoordination only) parentProcessId (long) (ProcessUnderCoordination only)

Task-based Engine Events

Event Name	Description	Event Properties
AeHumanTaskCreated	Fired when a WS-HT task is created. By the time this event fires, the task is created and available for querying.	TaskPAProcessId (Double) TaskPALocationId (Double) TaskId (String) TaskStatus (String) See also properties listed in "Activity States Event Properties Task States and Task Event Types " on page 437.
AeHumanTaskEvent	Fired when a WS-HT task changes state for any reason, such as when claimed, started, added, commented, output is submitted, or completed.	TaskEventType (String) TaskEventPrincipal (String) TaskPAProcessId (Double) TaskPALocationId (Double) TaskId (String) TaskStatus (String) See also properties listed in "Activity States Event Properties Task States and Task Event Types " on page 437.

Creating an Event-Action BPEL Process

An event-action process receives and handles the events you define in other processes.

To create an event-action BPEL process:

1.   In the Project Explorer, select the **File > New > BPEL Process**.

2.   Select a Workspace folder, type in a File name, and select Next.

3.   Select the BPEL template called Event Action Process.

A new BPEL file opens with a receive activity that takes an event trigger as input.

The event trigger is based on one of the following:

- Events you define in the eventing tab of the PDD of another process. In this case, the process containing event definitions must refer to the service name of the event action process.

- `AeEngineEngine` event. In this case, the service name you select in your PDD for My Role must be *AeEngineEvent*

- `AeHumanTaskCreated` event. In this case, your service name must be *AeHumanTaskCreated*.

- `AeHumanTaskEvent` event. In this case, your service name must be *AeHumanTaskEvent*.

A system event is consumed by the event action service for all running processes. However, you can narrow your event handling to particular processes by programmatically referring to a process, such as:

`$trigger/AeEngineEvent/processLocalName/text()='myProcess'`

The following is an example of an input message for the event-action process with a defined event:

```
<evt:trigger xmlns:evt="http://docs.active-endpoints.com/wsdl/eventing/2008/06/
eventing.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <orderitem xmlns:ext="http://dinner2">
```

```
<processId type="xsd:long">601</processId>
<eventId type="xsd:string">/process/flow
  /extensionActivity
  /peopleActivity[@name='PlaceOrder']:PID_601</eventId>
<timeStamp type="xsd:dateTime">2008-07-28T19:03:33.464Z
</timeStamp>
<planId type="xsd:int">19</planId>
<nodePath type="xsd:string">/process/flow
  /extensionActivity/peopleActivity[@name='PlaceOrder']
</nodePath>
<processName type="xsd:QName">ext:dinner</processName>
<state type="xsd:int">2</state>
<sessionId type="xsd:int">2</sessionId>
<locationId type="xsd:int">75</locationId>
<instanceNodePath type="xsd:string">/process/flow
  /extensionActivity/peopleActivity[@name='PlaceOrder']
</instanceNodePath>
<myOrderItemProp type="xsd:string">sandwiches
</myOrderItemProp>
</orderitem>
</evt:trigger>
```

Activity States Event Properties Task States and Task Event Types

Activity States, Event Properties, Task States, and Task Event Types

See the following tables for defining parameters for activity states, event properties, task states, and task event types.

Activity States

During execution, an activity or link is associate with a processing state. If you are defining an event, described in "Defining an Event in the Process Deployment Descriptor " on page 433, you must select one or more processing states to include in the definition.

The following is a description of each state that can be associated with an activity or link.

Activity State	Description
inactive	activity or link is not the next object to be executed and has not been executed
execute complete	normal completion of activity
dead path status	activity did not execute due to branch or link condition
suspended	Process is suspended on this activity
ready to execute	activity or link is the next object to execute
execute fault	abnormal completion of activity
terminated	Activity was terminated because either the process was terminated or one of the activity's ancestor containers faulted
faulting	A suspended process stopped on a faulting activity

Activity State	Description
executing	activity is executing normally
link status	true, if the link was executed

Event Properties

The following table describes the available properties that are part of the input trigger message.

Event Property	Type	Description
processId	long	Process ID of the BPEL process that generated the event
state	int	One of the state constants found in IAeProcessEvent, such as ReadyToExecute, Executing, ExecuteComplete. **Note:** See the table below for integer values.
eventId	string	Unique event id that is a combination of the location path and process ID of the activity that generated the event. This is useful, for example, to correlate a start event and completion event fired by the same activity.
instanceNodePath	string	Path to the activity that generated the event. This path includes instance information in the case that it is within a `forEach` or other container
nodePath	string	Path to the activity that generated the event. This path does not include instance information. It could be considered as the path into the process definition rather than the process instance.
locationId	int	ID of the activity that generated the event (similar to nodePath)
processName	QName	Name of the BPEL process that generated the event
timeStamp	Date	Time the event was fired
sessionId	int	Session ID of the process at the time the event was fired. The session ID is incremented whenever the process leaves memory for any reason.
faultName	string	Name of the current process fault, if one exists
planId	int	ID of the process plan associated with this process instance

One of the event properties is the *state* property. The following table describes the states and the integer value optionally available to use to refer to the state.

Event State	Description	Integer Value
READY_TO_ EXECUTE	The activity is now ready to execute	0
EXECUTING	The activity has begun executing	1
EXECUTE_COMPLETE	The activity has completed normally	2
EXECUTE_FAULT	The activity has faulted	3

Event State	Description	Integer Value
LINK_STATUS	A link has been evaluated	4
DEAD_PATH_STATUS	The activity has gone dead path (will not execute)	5
TERMINATED	The activity was terminated due to an Exit activity executing or an ancestor container faulting	6
SUSPENDED	The activity has been suspended	13
FAULTING	The activity is faulting	14
INACTIVE	The activity is inactive	-1

Task States

The following is a description of each task state that can be used as a string in the *TaskStatus* event property.

Task State (TaskStatus(string))	Description
READY	Task can be claimed by one of its potential owners
RESERVED	Once a potential owner claims the task, it transitions into the Reserved state, making that potential owner the actual owner
IN_PROGRESS	Task has been claimed and is being worked on
SUSPENDED	In any of its active states (Ready, Reserved, InProgress), a task can be suspended, transitioning it into the Suspended state.
EXITED	One of the final states (Completed, Failed, Error, Exited, Obsolete). The requesting application is no longer interested in any result produced by the task.
FAILED	Abnormal completion of task
COMPLETED	Successful completion of the work

Task Event Types

The following list includes valid strings for task event types:

activate	deleteFault	resume
addAttachment	deleteOutput	setFault
addComment	escalated	setGenericHumanRole
cancel	expire	setOutput
claim	fail	setPriority

complete	finalize	skip
create	forward	start
delegate	nominate	stop
deleteAttachmentById	reassign	suspend
deleteAttachments	release	suspendUntil
deleteComment	remove	updateComment

Process Central Forms and Configuration

The following topic discuss Process Central forms and configuration:

- What is Process Central?
-
-
-
-
-
-

What is Process Central

Process Central is a client application that contains process request forms, Tasks, and Reports. Users of Process Central can start a process, work on human tasks (from a BPEL for People process), and view reports related to tasks. In Process Developer, you create and deploy forms, tasks, and reports.

When the server is running, you can access Process Central in Process Developer at the URL:

```
http://localhost:8080/activevos-central
```

Note that your host:port can differ if you changed it during Process Developer installation.

At the **Login** page (only on the server running in Process Developer), you can enter the following username and password:

User name: **manager**

Password: **manager**

This user name is defined in the enabled Identity Service running on Process Developer's embedded server. By default, there is a `tomcat-users.xml` file enabled, and you can log in with this name or another of the user names listed that has the *abTaskClient* role. The `tomcat-users.xml` file is not enabled for Process Servers.

As a developer, you can contribute to Process Central as follows:

- Create a configuration file that customizes the navigation area of Process Central and describes the forms, tasks, and reports that are deployed.
  See "Creating a Process Central Configuration File " on page 449.

- Create a process request form so that users can start a process by submitting a form.
  See "Creating a Process Central Process Form " on page 442.

- Create a report
  See Creating Reports for Process Server and Central.

- Create an HTML rendering for a task in a People activity.
  See *Providing Renderings for Task Clients* in the Human Tasks section of this help.

The following illustration shows an example of Process Central. Note that the Tasks, Forms, and Reports bars on the left can slide up and down to change the display.

Creating a Process Central Process Form

Select an operation to create an HTML form used for sending a request message to a process. You can accept the default request folder and default file name for the process request form, or enter new ones.

For an overview of this topic, see What is Process Central?

A *process request form* is an HTML file generated from the input message for a starting (create instance) receive or pick activity. You deploy the form to the server, and users can then submit it to start a process from Process Central.

To create a Process Request Form:

1. We recommend that you create a process deployment descriptor (PDD) before creating a process request form so that the service you want to target is already known. If the service name (that is, the My Role service name in the PDD) is not known, you must add it manually to the form, as described in "Understanding the Process Request Form Template" on page 443.

2. Select **File > New > Process Request Form**.

3. Select the port type and operation defined in the starting activity of the process.

4. Accept the default location and filename for the form or modify them. The default filename is the operation name.

5. For multilingual support of the form, select **Advanced**. For details, see "Adding Multilingual Support to Process Central" on page 461.

See also:

*   "Understanding the Process Request Form Template" on page 443
*   "Editing HTML in Process Central Forms" on page 444
*   "Adding a New Service Operation for a Form or a Task " on page 445
*   "Adding a New Service Operation for a Form or a Task " on page 445
*   "Customizing Task and Form Scripts An Introduction " on page 446
*   "Using the Process Developer SDK for Customizing Forms " on page 448

Understanding the Process Request Form Template

When a user submits a process request form, the server must match the request to the service name in the PDD. Be sure that the service name shown in the form matches the My Role service name in the PDD (for the starting activity of the process).

In the generated form, find the line of code for the service name and verify the name:

```
<script type="text/javascript">
//<![CDATA[
  // request form to submit requests and handle response
   var AeRequestForm$ID = function() {
     // this
     var mFormInstance = this;
     // the service name
     var mServiceName = "EstimationService";
```

Making Additional Modifications

If you wish to make additional modifications to a process request form, it is helpful to understand the template used to create the form and the HTML file that is generated from it.

Visual Parts of the Form

*   **Title**
    By default the title is based on the operation name of the receive activity's interface. It is in the top-most table of the HTML page.

*   **Input Message Parts**
    If the message is a single-part element, there is one table containing one row for each child element. There is a separate table for each element. If an element is repeating, two command buttons are added: Add and Remove.

*   **Send Request command button**
    At the bottom of the form, a command button is added to submit the request and start the process.

To modify the form, see "Editing HTML in Process Central Forms" on page 444 and "Adding a New Service Operation for a Form or a Task " on page 445.

For multilingual support, see "Adding Multilingual Support to Process Central" on page 461.

Script Section of the Form

The script contains the commands to format and display the form dynamically in Process Central. The script is JavaScript, containing JavaScript Object Notation (JSON) and jQuery syntax to provide functionality in the form as well as transport protocol. For details, see "Customizing Task and Form Scripts An Introduction " on page 446.

Editing HTML in Process Central Forms

Use the Eclipse Web Page Editor for creating HTML Requests and Tasks for Process Central.

Once you create an Process Central form, it is displayed in the Web Page editor. You may want to make simple edits to the form as follows:

- Add instructions for end users.
  Add HTML tags, such as <p> </p>or <ul> to include details on how to use the form.

- Improve the look and function of the form with inline styles, images, and links.
  Add common HTML links such as <href> and <img>. Images can be deployed to the Process Server as resources within the business process archive (BPR) file. For details, see "Deploying Forms Reports and Configuration Files" on page 460.

- Add multilingual support (internationalization) that is not added in the Web Page Editor. See "Adding Multilingual Support to Process Central" on page 461.

- Add additional service operations to the form for multiple inputs and/or outputs. See "Adding a New Service Operation for a Form or a Task " on page 445.

About the Web Page Editor

Process Developer includes the Eclipse Web Page editor for editing request and task forms. Note that in Process Developer Preferences, a setting is enabled for the Web Page editor to include a Process Central style sheet. The style sheet includes the font and background styles for Process Central.

Editing in the Web Page Editor

The Web Page editor contains two editing panes: Design and Source. The Design pane shows the form in graphical mode and the Source pane contains the HTML tags and JavaScript functions.

Toolbar buttons allow you to show or hide either pane or split the panes horizontally or vertically. Additionally, you can apply simple formatting for text elements, such as bold or italics.

An Outline view is helpful for selecting a construct in the editing pane.

For a view of what the form will look like in Process Central, select the Preview tab. Preview shows the placement and look of paragraphs, tables, control buttons and other constructs, but command buttons are for display only.

In Design mode, use the HTML palette to add additional constructs, such as text fields, images and lines.

See also:

- "Adding a New Service Operation for a Form or a Task " on page 445
- "Customizing Task and Form Scripts An Introduction " on page 446
- "Using the Process Developer SDK for Customizing Forms " on page 448
- "Testing and Debugging Process Central Forms " on page 449

Adding a New Service Operation for a Form or a Task

Select an operation to create an HTML form used for sending a request message to a process. You can also select an operation to add a new service to a process request or task form.

If desired, you can modify an existing form or task form by adding fields and functions generated from service operations. A service operation is a WSDL operation from a Workspace WSDL that represents the endpoint of a BPEL process running in A Process Server. The operation must have a service name associated with it.

When a task or form is open in the Web Page editor, the Participants view, normally displaying process participants, shows only the service operations available for forms, as shown.
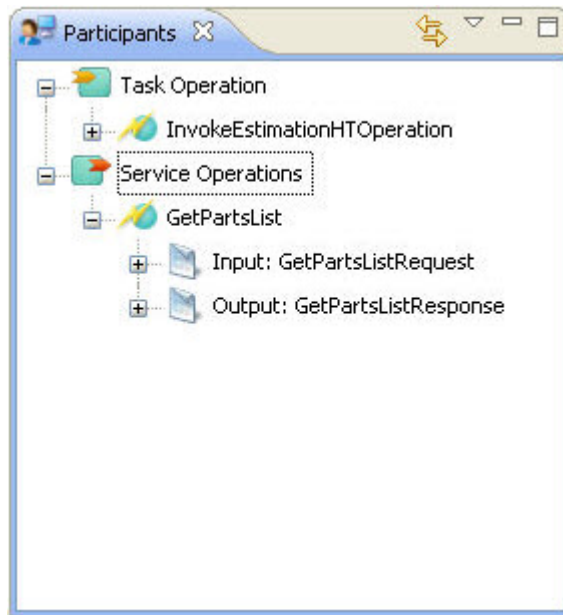
There are several use cases for adding new service operations. Here are some examples:

- If you have an existing task or form, but the schema for the task or service operation has been updated, you can modify or add the additional fields to your existing form by dragging and dropping them into the form

- Enhance the functionality of form fields by calling a service. For example, if the form field is "Select a Country," call a service to provide a list of country names.

If you have a deployed process that contains an operation you want to use, you must import the WSDL service definition into your workspace.

To add a WSDL to the Workspace:

1. In the Process Console, display the Catalog Service Definitions page.

2. Select the Service Definition of a deployed process to display the WSDL for the service.

3. Copy the URL. For example, `http://localhost:8080/active-bpel/services/loanservice?wsdl`.

4. In your project, right-mouse click on Service References.

5. Select **Import**.

6. Paste the URL into the field for Enter the URL to import from, and select Finish.

To add a new service operation:

1. Open a task or form in the Web Page editor.

2. In the Participants view, right-mouse click on Service Operations, and select **Add Service Operation**.

3. In the Select Service Operation dialog, select a Port Type and Operation.

4. Notice that the service name may be unknown, if it does not exist in the process PDD file. Ensure that you add the service name, as described in "Understanding the Process Request Form Template" on page 443. If the name is unknown, the dialog lets you know, as shown:

Events that happen when you add a new service operation:

- A **Send Request** button is added to the bottom of the HTML form for the new operation

- A new JavaScript section is appended to the source file. This is the section where you must specify the service name.

- The operation and input/output messages are added to the Participants view

To use the new service operation in your form:

1. Ensure that your form is in focus in the Web Page editor.

2. In the Participants view, expand an operation to display the input and/or output message.

3. Drag the operation's input/output message, or a message's part, or a part's element to a location in the Design view.

If you are adding multilingual support for forms, you must manually add multilingual support to the new fields and add the language values to properties files. For details on how to manually update the form, review "Adding Multilingual Support to Process Central" on page 461.

See also:

- "Editing HTML in Process Central Forms" on page 444

- "Customizing Task and Form Scripts An Introduction " on page 446

- "Using the Process Developer SDK for Customizing Forms " on page 448

Customizing Task and Form Scripts An Introduction

The generated HTML forms for tasks and forms fulfill a basic requirement of rendering XML message data into HTML. In addition, the form allows for powerful customization by taking advantage of several HTML-based technologies that Process Developer uses to create the form, namely:

- JavaScript and Asynchronous JavaScript and XML (AJAX)

- JavaScript Object Notation (JSON)

- JQuery and JQuery UI

These technologies in combination provide dynamic forms and tasks. Note the following development features for a form:

- Uses a standards-based solution

- Allows you to work in any HTML editor

- Provides many rendering possibilities for task and form message data, such as date pickers, tabs, check boxes and combo boxes
- Allows rich additions, including calls to additional services, images, and most anything you can script for a Web application project
- Provides access to large amount of technical resources, knowledge bases and web application developers

Below are some details on each of the technologies used to create tasks and forms.

JavaScript and AJAX

JavaScript is the binding for the form and includes the subset of JSON and JQuery. AJAX calls services asynchronously in the background without interfering with the display and behavior of forms and tasks.

JSON

JavaScript Object Notation (JSON) is a data interchange format. XML messages from the form or task are converted to a JSON object. Process Developer's implementation of JSON is within the Google Data Protocol. For more information on JSON, see `http://code.google.com/apis/gdata/json.html`.

The Google Data Protocol has the following rules:

- XML is represented as a JSON object.
- XML version and encoding attributes are converted to version and encoding attributes of the root element, respectively.
- If an element has a namespace alias, the alias and element are concatenated using "$". For example, ns:element becomes `ns$element`.
- Each nested element or attribute is represented as a name/value property of the object.
- Attributes are converted to string properties.
- Child elements are converted to object properties.
- Elements that can appear more than once are converted to array properties.
- Text values of tags are converted to `$t` properties.

The following is an example of a XML to JSON conversion:

XML Message:

```
<?xml version="1.0" encoding="UTF-8"?>
<types1:EstimationRequest
   xmlns:ns="http://example.org/QuoteRequest.xsd"
   xmlns:ns1="http://example.org/QuoteRules.xsd"
   xmlns:types1="http://example.org/Estimation.xsd">
  <ns:refNumber>1</ns:refNumber>
  <ns:quoteRequest>
    <ns:contact>
      <ns:name>JZ</ns:name>
      <ns:email>activevos@gmail.com</ns:email>
    </ns:contact>
    <ns:model>500</ns:model>
    <ns:year>1983</ns:year>
    <ns:description>a fine car</ns:description>
  </ns:quoteRequest>
</types1:EstimationRequest>
```

JSON Object:

```
{"EstimationRequest":{"xmlns":"http:\/\/example.org\
    /Estimation.xsd",
 "refNumber":{"xmlns":"http:\/\/example.org\
     /QuoteRequest.xsd","$t":"1"},
 "quoteRequest":{"xmlns":"http:\/\/example.org\
     /QuoteRequest.xsd",
```

```
"contact":{"name":{"$t":"JZ"},
"email":{"$t":"activevos@gmail.com"}},
"model":{"$t":"500"},
"year":{"$t":"1983"},
"description":{"$t":"a fine car"}}}}
```

Tip: In order for forms to work with Process Central (JSON + JavaScript), the form element names must observe the same rules used to define JavaScript variables. For example, the variables can only contain contains letters, numbers, or underscores. Also element names should not be a reserved JavaScript keyword.

jQuery and JQuery UI

jQuery is a JavaScript library that simplifies HTML document traversing, event handling, animating, and Ajax interactions. jQuery is built to select certain elements on a page and manipulate them. jQuery UI provides visual controls, including core interaction plug-ins as well as many UI widgets.

For details, see `http://jquery.com/`

JQuery Example

The selection process in jQuery is a filter process, whereby every element on a page is put through the filter you supply in a command, and the command returns either the single matching object or an array of matching objects from which you can traverse.

For example, to get an array of all the matching HTML elements in a page, you pass the HTML tag into the jQuery search field:

```
// Create a red background for every <p> tag in the page.
$("p").css("background", "#ff0000");
```

In a form, Process Server uses jQuery to select data elements. In the following example, a jQuery command creates a table for repeating elements. If a user wants to add a new row, the following code is used:

```
// Handles the adding of a new repeating field
$("#processRequestForm$ID").find("table.avc_repeating_field_data_table
input[type='button'].avc_add_field_element").click(function() {
   var table = $(this).parents("table:first");
   var rows = $(table).find("tbody tr").length;
   var lastRow = $(table).find("tbody tr:last");
   var newRow = $(lastRow).clone(true);
   $(lastRow).after(newRow);
```

HTML element ID attributes must end with `$ID` suffix, such as in the example above (`#processRequestForm $ID`). The `$ID` suffix provides a unique HTML element ID to avoid collisions with other forms.

See also:

Using the Process Developer SDK for Customizing Forms

To take full advantage of customizing forms and tasks, we recommend that you download and familiarize yourself with the Informatica Business Process Manager SDK Package. This package includes a detailed discussion of Process Developer XML-JSON and Process Developer XML-JSON for Process Central, showing you form development and usage. (This discussion is also part of this help.)

Where to find the Process Developer SDK

The SDK Package and Documentation kit is available in the Informatica InfoCenter. The latest InfoCenter is located at `http://infocenter.activevos.com`. Note that you must use the version of the SDK that is available for your product version.

You can link remotely to the InfoCenter's SDK documentation from Process Developer Online Help. For instructions on displaying remote help, refer to the Help Preference in Process Developer.

Testing and Debugging Process Central Forms

After you deploy one or more process request forms and task forms, you can test and debug your JavaScript by using a browser JavaScript Debugger and enabling debugging in Process Central. You can also use the JSON Converter utility.

To debug your JavaScript in HTML forms, do the following:

- Install or enable a JavaScript debugging tool in your browser, such as Firebug for Firefox, built-in debugging in Internet Explorer, or the Developer JavaScript Debugger in Chrome.
- To enable debugging in Process Central, add the following command to the Process Central URL as follows:
  `http://localhost:8080/activevos-central/avc/avc.jsp`**?debug=true**
- Use the **JSON Converter** tool to check the conversion of JSON to XML and vice versa. Check what a JSON request form looks like in XML or generate sample XML data and convert to JSON. Access the converter at:
  `http://localhost:8080/active-bpel/jsonConverter.html`

When you make changes in your form and redeploy it, you can select **Reset Form** in Process Central to view the updated form.

With debug enabled, it is easier to debug jQuery functions. They get expanded into readable lines of code instead of the one long line of code that typically shows in a debug console.

For task forms, you can also use Development Mode for testing layouts without redeploying the form.

Tips for testing forms:

- Be sure that your HTML forms are well-formed XML. Process Central validates forms and provides warnings if there are invalid tags.
- Be sure to use lower-case element names that are schema-valid.
- Check that all tags are properly closed with end tags.
- In HTML markup, mixed types are not supported. For example, within an XML message element, you cannot add HTML tags, such as `<contactName><b>Georgia Jane</b></contactName>`.
- HTML element ID attributes must end with `$ID` suffix, such as `processRequestForm$ID`. The `$ID` suffix provides a unique HTML element ID to avoid collisions with other forms.
- JavaScript reserved words, such as `in` and `out` are not allowed as JSON data parts. If your schema contains an element named as a JavaScript reserved word, you may see a blank form in some browsers. To correct the problem, you must change the JavaScript portion of your form as shown in the example:
  Incorrect: getText(parametersTaskInputPart.**NewOperation.in**) where *in* is an input message for NewOperation.
  Correction: getText(parametersTaskInputPart.**NewOperation["in"]**)

Creating a Process Central Configuration File

The Process Central Web application contains Forms, Tasks, and Reports that you configure for navigation.

A basic configuration file resides in Process Server to provide the initial configuration for navigation filters. You can view the file from the Catalog Resources page of the Process Console.

You can create additional configuration files that customize the navigation filters, on a user by user or group by group basis, and also describe the forms and reports to be deployed.

You can also replace the basic configuration file with your own file in Process Server. All elements from the basic configuration file are required, and by default the basic file is deployed. Your modifications override the basic settings.

To create an Process Central Configuration (.avcconfig) file

1. Select **File > New > Central Configuration**.

2. Accept the default location and filename for the file or modify them. The default location is the deploy folder, and the filename is based on the project name.
   The file opens in an XML editor, and the main sections of the file are commented out. See Editing Tips below for help in editing the file.

Overview of Configuration File Sections:

Section	Configuration filters you can add
Task Role Filters: See "Configuring Task Role Filters" on page 453	- Items in the Role drop-down - Items in the Show drop-down - Folder names in the Tasks list - Show or hide folders based on users and groups (allowed roles) - Custom column names for tasks - Task list filtering
Forms: See "Configuring Form Filters" on page 451 Reports: See "Configuring Reports Filters" on page 452	- Folder names in the Forms list or Reports list - Show or hide folders based on users and groups (allowed roles)
Localization	See "Adding Multilingual Support to Process Central" on page 461
Contributions to Process Central, Forms and Tasks	Create your own scripts, CSS, and other elements. Specify them in the <centralIncludes> section. For details, see "Including Your Own Styles Scripts and Meta Data for Process Central" on page 452.

Editing Tips:

- Uncomment the section you want to work on, such as the Task Role Filters section.

- For readability of your file, delete the sections you do not want to deploy in this file and create separate configuration files for each of the sections.

- Forms must be valid XML so be sure to close HTML tags.

- Use Content Assist (CTRL + spacebar) to add new elements to the file based on the Process Central configuration schema.

- In the Outline view, right-mouse click on an element to add context-sensitive attributes and children

- You can view the avoscentral-config.xsd schema in your Process Developer installation at:
  `<installdir>designer\plugins\com.activee.avoscentral.services_N.N\schema`

- You can view the base avcconfig file from the Catalog Resources page of the Process Console from which you may want to copy sections to include in your file.

Configuring Form Filters

For an overview of form, see "Creating a Process Central Process Form " on page 442.

The Forms section of the Process Central configuration file configures the following:

- Folder name to add under the Forms category
- Fork name to add to the folder
- Description of the form to display in the work area
- Location and name of the form to be deployed
- Restricted access to the form for named user roles

Here is an example of a configuration in Process Central showing a custom folder containing two forms.
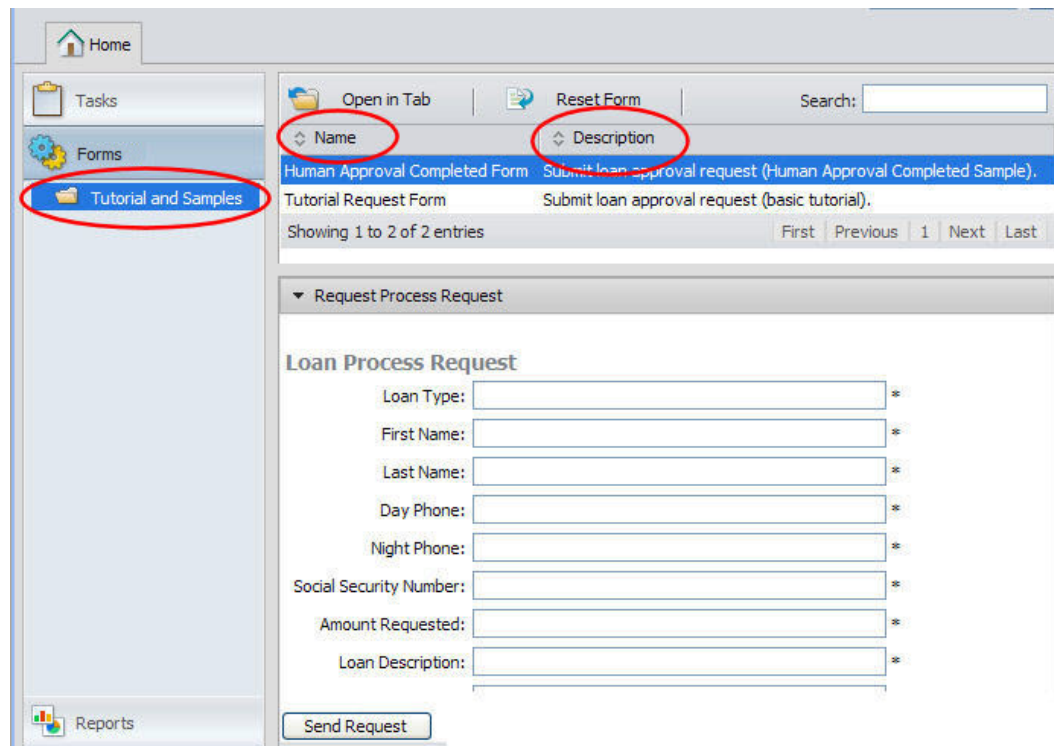
Fill in the values as shown in the following example:

```
<tns:requestCategoryDefs>
  <tns:requestCategoryDef id="loans_category"
   name="Loan Approval">
    <avccom:requestDef id="loan_request"
     name="Customer Request Form">
<!-- Use this section only to restrict access to requests.
    Delete it if not used.
    <avccom:allowedRoles>
      <avccom:role>FILL_IN_ROLE_1</avccom:role>
      <avccom:role>FILL_IN_ROLE_2</avccom:role>
    </avccom:allowedRoles>
-->
    <avccom:description>
       Submit a request for loan approval.
    </avccom:description>
      <avccom:formLocation>
        project:/tutorial/form/request/loanRequest.html
      </avccom:formLocation>
    </avccom:requestDef>
```

```
        </tns:requestCategoryDef>
      </tns:requestCategoryDefs>
```

Configuring Reports Filters

You can design Business Intelligence and Reporting Tools (BIRT) reports in Process Developer and deploy them for display in Process Central as well as in the Process Console. For details, see Creating Reports for Process Server and Central.

The Reports section of the Process Central configuration file configures the following:

- Folder name to add under the Reports category.

- Report name to add to the folder.

- Description of the report to display in the work area.

- Location and name of the report to be deployed.
  Tip: If you want to view some standard reports, look in the Process Console.

- Location and name of an optional HTML file created for the Preview area.

- Restricted access to the report for named user roles.

Fill in the values as shown in the following example:

```
<tns:reportCategoryDefs>
  <tns:reportCategoryDef id="loans_category"
              name="Loan Approval">
    <avccom:reportDef id="loan_report"
              name="Loan Approval Report">
<!-- Use this section only to restrict access to reports.
    Delete it if not used.
        <avccom:allowedRoles>
            <avccom:role>FILL_IN_ROLE_1</avccom:role>
            <avccom:role>FILL_IN_ROLE_2</avccom:role>
        </avccom:allowedRoles>
-->
        <avccom:description>
            Loan Application Approvals By Month
        </avccom:description>
        <avccom:formLocation>
            project:/tutorial/reports/loansReport.rptdesign
        </avccom:formLocation>
        <avccom:formPreview>
            project:/tutorial/form/preview/loanRptPreview.html
        </avccom:formPreview>
    </avccom:reportDef>
  </tns:reportCategoryDef>
</tns:reportCategoryDefs>
```

Including Your Own Styles Scripts and Meta Data for Process Central

In a new `avcconfig` file, there is a section where you can specify your own CSS files, scripts, and meta data for use in Process Central.

Instead of linking in CSS, scripts, and meta data in form and tasks, you reference it from a `.avcconfig` file in the `<centralIncludes>` section.

The `<centralIncludes>` section can include the following elements that are common `<head>` element in a XHTML file:

- **<script>**. Add a reference to a JavaScript or other script that contains the functions you want to use in request and task forms. The syntax in `<centralIncludes>` is, for example:
  `<script xmlns="http://www.w3.org/1999/xhtml" | type="text/javascript" src="myJquery.script.js"/>`

  You can also use inline scripts, such as `<script xmlns="http://www.w3.org/1999/xhtml" type="text/javascript"> var my_String = "Hello";`

- **<meta>**. Add meta data keywords. The syntax is, for example:
  `<meta xmlns="http://www.w3.org/1999/xhtml" name="keywords" content="some keyword,another keyword" />`

- **<style>**. Use inline styles, such as `<style xmlns="http://www.w3.org/1999/xhtml" type="text/css">p {background-color: yellow}</style>`

- **<link>**. Style sheet link. See example below.

When you create a new `avcconfig` file, by default, only one section of the file is generated:

```
<tns:centralIncludes>
<link xmlns="http://www.w3.org/1999/xhtml" href="../example.css" rel="stylesheet"
type="text/css">
</tns:centralIncludes>
```

To use your own style sheet:

1. If desired, review the default Process Central CSS files. To do this, open Process Central and view the Page Source. Find the section of the source file linking to the CSS files and select a link. The style sheet for basic UI styles for fonts and colors is *ae-avc.css*.

2. If desired, copy styles from `ae-avc.css` or other CSS and make your own modifications.

3. In forms and tasks, use the class names from the CSS file, as desired.

4. Deploy your CSS file.

5. In an `avcconfig` file, add the name of the CSS in the <tns:centralIncludes> section and deploy the `config` file. For example:

   ```
   <link xmlns="http://www.w3.org/1999/xhtml"
       href="../example.css" rel="stylesheet" type="text/css">
   ```

   Where the `href` path can be one of the following:

- relative path to this `avcconfig` file within your project

- full `project:/path/to/file`

- Absolute URL, such as `http://domain:port/myCSS.css`

Configuring Task Role Filters

The Task Role Filters section of the Process Central configuration file allows for comprehensive configuration of the Tasks view, as shown by the circled areas on the following illustration.
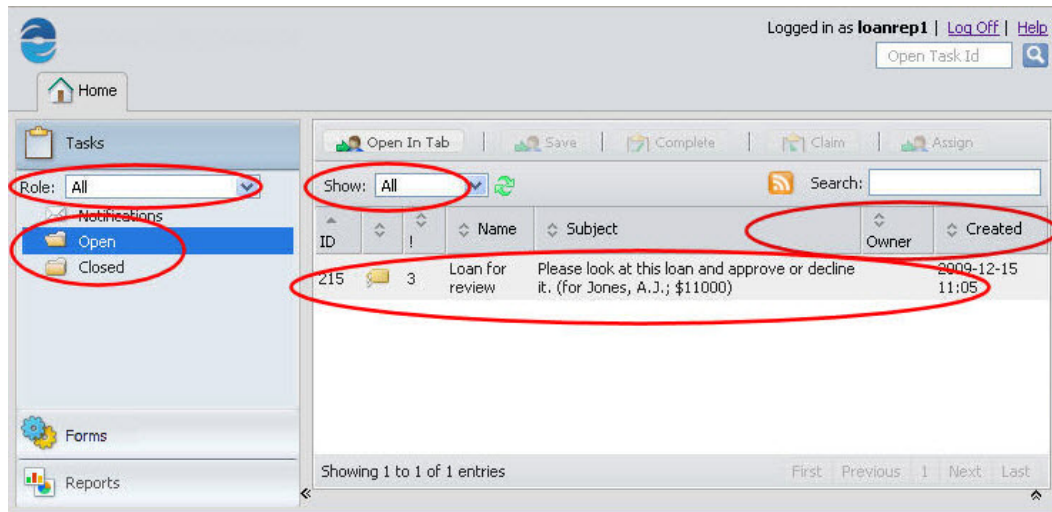
You can add basic and advanced configuration.

Basic Configuration

- For details, see "Basic Task Roles Filter Configuration " on page 454

- Basic configuration requires only string replacement in the `.avcconfig` file for the following:

- New role names to add to the Roles list in the Task list to replace the basic WS-HT names of User, Administrator, Initiator, and Stakeholder

- New folder categories to replace or add to the list of Notifications, Open Tasks and Closed Tasks

- Users and groups from your Identity service allowed to view the new categories (not necessarily related to role assignments in a People activity)

- New task lists to add to the Show list to replace or add to All, Claimed and Unclaimed

Advanced Configuration

Advanced configuration provides customized task display and searching, including:

- Custom columns for tasks. For details, see "Configuring Custom Columns for Task Roles Filtering " on page 456

- Comprehensive filtering of a task list. For details, see "Configuring a GetMyTasks Filter with a WhereClause " on page 457.

- RSS feed filters for a task list. See "Configuring an RSS or Atom Feed Filter " on page 459.

Basic Task Roles Filter Configuration

For an overview of this topic, see "Configuring Task Role Filters " on page 453.

In the Process Central configuration file (`.avcconfig` file), you can modify the default configuration in the Task Role Filters section as follows.

To add a new role to the Role list:

The base set of roles include the WS-HT roles of user (potential owner), administrator (business administrator), initiator (task initiator) and stakeholder. You may have your own role names to add, based on the WS-HT roles.

In the following line of the file, add an id of your choosing and fill in a name attribute for the Role name to be displayed.

```
<tns:taskRoleFilterDef id="FILL_IN_ROLE_NAME" name="FILL_IN_ROLE_NAME">
```

To add a new item to the Show list:

The base set of Show items displayed includes All, Unclaimed and Claimed.

In the following line of the file, add an Id and fill in a name attribute for the filter name to be displayed in the Show list.

```
<avccom:taskFilterDef id="FILL_IN_TASK_FILTER_ID" name="FILL_IN_TASK_FILTER_NAME">
```

To add a new category folder to the Task list:

The base set of categories includes Notifications, Open tasks and Closed tasks.

In the following line of the file, add an Id and fill in a name attribute for the category name to be displayed in the Task list.

```
<tns:taskFilterCategoryDef id="FILL_IN_FILTER_CATEGORY_ID"
name="FILL_IN_FILTER_CATEGORY_NAME">
```

To show or hide folders for specified users and groups:

You can specify which users and groups can view the new folders you add. Using the Identity Service enabled in Process Server, copy the Role names associated with users and groups. If all users should see the new folders, remove the `<role>` elements.

Do not leave a `<role>` element as shown in the default, and do not leave the element empty because it means no one is allowed to see the folders.

```
<avccom:allowedRoles>
   <avccom:role>FILL_IN_ROLE_1</avccom:role>
   <avccom:role>FILL_IN_ROLE_2</avccom:role>
</avccom:allowedRoles>
```

You must delete the `<roles>` if you do not intend to specify user roles. The default value means that a user must be in the FILL_IN_ROLE_1 group to view the folder.

Example Configuration:

```
<tns:taskRoleFilterDefs>
  <!-- for All users in the Role picklist -->
  <tns:taskRoleFilterDef id="All_Role" name="All">
      <!-- New folder -->
    <tns:taskFilterCategoryDef id="Loan_Apps" name="New Car Loans">
         <!--  Display the folder to these users and groups -->
        <avccom:allowedRoles>
            <avccom:role>group_Auto</avccom:role>
            <avccom:role>gDriver</avccom:role>
        </avccom:allowedRoles>
        <!--  Add this filter to the Show picklist -->
        <avccom:taskFilterDef id="auto_loans" name="InProgress">
    </tns:taskFilterCategoryDef>
  </tns:taskRoleFilterDef>
</tns:taskRoleFilterDefs>
```

For advanced configuration, see the following:

-
-
-

Configuring Custom Columns for Task Roles Filtering

In the Task Role Filters section of the `.avcconfig` file, you can add the task column names you want to display for a category (folder) of tasks. There is a default configuration of column names, and you can add to or replace the columns displayed.

- To add standard WS-HT column names, see WS-HT Task Property List elsewhere in this help

- To add custom column names see Creating Custom Task Properties elsewhere in this help

- To localize the column names, see the example below

Default Configuration

The default configuration for `taskColumn` uses standard WS-HT task properties (all prefixed with *Task*):

```
<avccom:taskColumns>
    <avccom:taskColumn property="Task.Id">Id</avccom:taskColumn>
    <avccom:taskColumn property="Task.Status"></avccom:taskColumn>
    <avccom:taskColumn property="Task.Priority"></avccom:taskColumn>
    <avccom:taskColumn property="Task.Name"></avccom:taskColumn>
    <avccom:taskColumn property="Task.PresSubject"></avccom:taskColumn>
    <avccom:taskColumn property="Task.CreatedOn"></avccom:taskColumn>
</avccom:taskColumns>
```

To replace or add to these columns, you can add your own column specification, using a custom task property, such as:

```
<avccom:taskColumn property="first">First Name</avccom:taskColumn>
```

You must create the custom column names as part of the task definition in a People activity.

To localize the column name, you can specify a key from a message properties file that you create. For example, the column name below is specified as `${bundle.key}`:

```
<avccom:taskColumn property="Task.CreatedOn"
    type="dateTime">${bundle.key}</avccom:taskColumn>
```

For details on creating a message properties file, see <u>"Providing Multilingual Support for Task Forms" on page 534</u>.

The following sample Task Role Filter section illustrates the use of custom columns for Process Central.

```
<!--
    Example - adding a new category/folder called Hello Tasks.
    This folder will appear in the All role.
 -->
<tns:taskRoleFilterDefs>
  <tns:taskRoleFilterDef
      id="ae-avc-role-all" name="${ae.avc.role.user.name}">
    <tns:taskFilterCategoryDef id="custom-cat"
        name="Hello Tasks">
      <avccom:allowedRoles>
      </avccom:allowedRoles>
      <!--
        Define custom columns that appear in Process Central
        (or remove/comment out root element to use
        default column list provided out of the box).
      -->
      <avccom:taskColumns>
          <!-- Task.Id, Task.Status and Task.Name are
               standard WS-HT columns.
          -->
        <avccom:taskColumn
             property="Task.Id">Id</avccom:taskColumn>
        <avccom:taskColumn
             property="Task.Status"></avccom:taskColumn>
        <avccom:taskColumn property="Task.Name">Task Name</avccom:taskColumn>
        <avccom:taskColumn property="Task.PresName">
             Display Name</avccom:taskColumn>
           <!-- You can also specify the column data
```

```
                              formatting via the type attribute. Useful
                              values are date, dateTime and boolean. The
                              column display name can also be externalized
                              via ${18n.key} method.
                     -->
              <avccom:taskColumn property="Task.CreatedOn"type="dateTime">
                     ${bundle.key}</avccom:taskColumn>
                <!-- The property names 'first' and 'last' are
                     WSHT task presentation parameters defined in
                     your tasks (custom properties).
                 -->
              <avccom:taskColumn property="first">First Name</avccom:taskColumn>
              <avccom:taskColumn property="surname">Surname (Last Name)</avccom:taskColumn>
         </avccom:taskColumns>
         <avccom:taskFilterDefRef ref="ae-avc-user-open-all"/>
         <avccom:taskFilterDefRef
                             ref="ae-avc-user-open-unclaimed"/>
         <avccom:taskFilterDefRef
                             ref="ae-avc-user-open-claimed"/>
         <avccom:taskFilterDefRef
                             ref="ae-avc-user-open-suspended"/>
       </tns:taskFilterCategoryDef>
    </tns:taskRoleFilterDef>
  </tns:taskRoleFilterDefs>
```

Configuring a GetMyTasks Filter with a WhereClause

In the Task Role Filters section of the `.avcconfig file`, you can add a filter to apply to the tasks in one of the Show pick entries. For example, you can create a filter for *Unclaimed Tasks* to show only Priority zero (highest priority).

Filtering a Task List using a Show Filter

Modify the default configuration in the `<avccom:filter/>` section of the `<avccom:taskFilterDef/>` section.

The following code snippet shows the default configuration:

```
<avccom:filter>
  <tsst:getTasks xmlns:tsst="http://schemas.active-endpoints.com/b4p/wshumantask/
2007/10/aeb4p-task-state-wsdl.xsd">
   <htdt:getMyTasks
        xmlns:htdt="http://www.example.org/WS-HT/api/xsd">
     <htdt:taskType>TASKS</htdt:taskType
     <htdt:genericHumanRole>POTENTIAL_OWNERS</htdt:genericHumanRole>
     <htdt:status>READY</htdt:status>
     <htdt:whereClause>Task.Name =
             'ApproveLoan'</htdt:whereClause>
     <htdt:maxTasks>20</htdt:maxTasks>
   </htdt:getMyTasks>
  <tsst:taskIndexOffset>0</tsst:taskIndexOffset>
  </tsst:getTasks>
</avccom:filter>
```

Filtering a Task List by Parameter using a WS-HT whereClause

You can use the following operators in the whereClause:

- >
- >=
- <
- <=
- =

- like

- and

- or

- ()

The following examples show various `whereClause` syntax. Note that the Booleans `and` and `or` are case-sensitive.

For a complete list of task column names, task types, generic human roles and other WS-HT specifications, you can work with the Informatica Business Process Manager SDK.

Example 1

The following example shows a sample `whereClause` using a standard WS-HT task property name.

```
<htdt:whereClause>Task.Name = 'ApproveLoan'</htdtwhereClause>
```

Example 2

The following example combines filter properties:

```
Task.Priority &lt; 3 and Task.Owner = 'loanrep1'
```

Example 3

The following example also combines filter properties.

```
Task.PaProcessId = 10
     and(Task.Escalated = true
     or  Task.Priority = 0)
```

Example 4: Using Presentation Parameters (Custom Task Properties)

The following example shows using presentation parameters:

```
Task.Name = 'LoanApproval'
     and loanAmount &gt; 10000
     and zipCode = '06484'
```

Presentation Parameters

If you query on presentation parameters, additional information is returned.

```
<tsst:propertyValues
     xmlns:tsst=
       "http://schemas.active-endpoints.com/b4p/wshumantask/
          2007/10/aeb4p-task-state-wsdl.xsd">
   <tsst:propertyValue
       name="parentprocessid" type="ns:string"
       xmlns:ns="http://www.w3.org/2001/XMLSchema">5013
   </tsst:propertyValue>
</tsst:propertyValues>
```

If no task is deployed with the presentation parameter, an error message is returned:

```
<htdt:illegalArgument
   xmlns:htdt="http://www.example.org/WS-HT/api/xsd">
     Task parameter property 'parentProcessId1' does not
      exist.
</htdt:illegalArgument>
```

However, if no task with this parameter exists but it is a valid name, the returned information is empty. Here is an example of this kind of parameter:

```
<aetgt:getMyTasksResponse
     xmlns:aetgt="http://www.example.org/WS-HT/api/xsd"
     xmlns:htdt="http://www.example.org/WS-HT/api/xsd"
     xmlns:result="http://saxon.sf.net/xquery-results"
```

```
xmlns:tss="http://www.activebpel.org/b4p/2007/10/wshumantask/aeb4p-task-storage.wsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
```

This element lists tasks that a `parentProcessId` parameter and also returns what this value is:

```
<xsd:getMyTasks>
    <xsd:taskType>TASK</xsd:taskType>
    <xsd:whereClause>parentProcessId >''</xsd:whereClause>
</xsd:getMyTasks>
```

The following table summarized what can occur when you use a presentation parameter.

Presentation Parameter	Is it Deployed	Actual Values	Query	Result
MyString	Yes	"a", "b"	*<a where clause>*	*<a parameter result>*
MyStringB	Yes	No Values	*<a where clause>*	*<a parameter result>*
MyStringC	No	Not applicable	*<a where clause>*	*<a parameter result>*

Using a getMyTasks orderBy Element

You can sort the results returned by `getTasks` by using the `orderBy` element. The fields name you can use are:

- Created
- Expiration
- Modified
- Name
- Owner
- PresentationName
- PresName
- PresSubject
- Priority
- State
- Summary

The following example sorts the results with the most recent first (that is, in descending order), followed by the priority:

```
OrderBy orderBy = new OrderBy();
orderBy.getFieldId().add("-Created");
orderBy.getFieldId().add("-Created");
```

Notice the minus sign ("-") to set the order to descending.

In XML, this same example is:

```
<ns:orderBy>
    <ns:fieldId>-Created</ns:fieldId>
    <ns:fieldIPdriority</ns:fieldId>
</ns:orderBy>
```

Configuring an RSS or Atom Feed Filter

For an overview of this topic, see <u>"Configuring Task Role Filters " on page 453</u>.

In the Task Role Filters section of the `.avcconfig` file, there is an element containing default URL details for a RSS or Atom Web feed. Process Central contains a Web feed subscription service that users can select to be notified about task updates.

In the `.avcconfig` file, the default query and format is shown as:

```
<avccom:feedQueryString format="rss" />
```

At runtime, the Web feed URL is automatically generated based on `<getMyTasks>`. For example, in Process Central, if the Tasks list is showing *open* tasks, the feed will send updates when new tasks arrive.

The default Web feed format is rss, but you can change it to *atom*, if desired.

If you leave the Task Role Filters section commented out when you deploy the `avcconfig` file, the default Web feed URL is generated from the default `avoscentral.avcconfig` file. You can look at this file in the Process Console, under the Catalog section.

You can modify the Web feed query filter as follows:

- Configure the taskFilterDef section. Your configuration details are automatically used when the feed query is generated. The following example shows that the `feedQueryString` is enclosed as part of the filter definition. You do not need to modify the string.

```
<avccom:taskFilterDef >
   <avccom:filter>
    <tsst:getTasks ...>
     <htdt:getMyTasks>... </htdt:getMyTasks>
    </tsst:getTasks>
   <avccom:feedQueryString format="rss" />
   </avccom:filter>
</avccom:taskFilterDef>
```

- (Not recommended). We do not recommend that you modify the default `feedQueryString`. Each display in Process Central automatically generates the Web feed URL based on the page's filters. There should not be a need to modify this. However, you can modify the `feedQueryString` by adding the well-known parameters from the WS-HT `getMyTasks` operation. For example:

```
<avccom:feedQueryString format="rss">filter=reserved</avccom:feedQueryString>
```

- For a detailed list of parameters, refer to *Getting Task Lists As A RSS or ATOM Feed* in the *Process Developer WS-HumanTask API* help.

Deploying Forms Reports and Configuration Files

Files that you want to deploy to the Process Server for use in Process Central, include Process Central Configuration files, HTML files for forms, BIRT report designs, and related resources, such as images and multilingual properties files.

You can create a business process archive (.BPR) file that includes only these resources.

To create a BPR, select **File > Export > Orchestration > Contribution-Business Process Archive**. On the Catalog Resources page, select the following Resource types:

- **Central Configuration** for Process Central configuration file
- **HTML** for forms
- **Binary** files for images
- **Property files** containing externalized strings for languages other than English. If these file are not already displayed, add them as type **Other**.
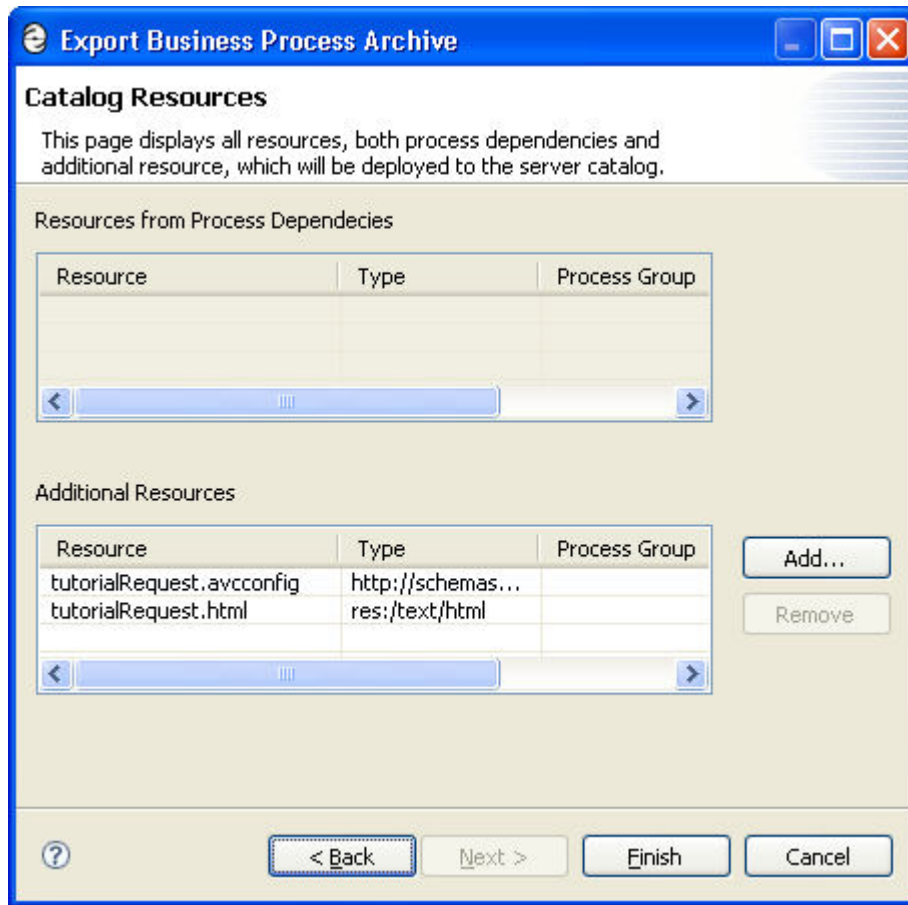
See also:

- "Deploying Additional Resources " on page 376.
- "Creating a Process Central Configuration File " on page 449
- "Creating a Process Central Process Form " on page 442
- For details about creating Human Task forms (Task Renderings), see *Human Tasks* within the *Process Developer Help*.

Adding Multilingual Support to Process Central

You can create different language versions of Process Central as well as the forms and reports that are deployed to it. The Process Central application user interface can display in a Web browser's preferred language. There is also language and locale support for requests, tasks, and reports.

The basic approach for multilingual support is to externalize strings. For each supported language, you use an existing or create a new default properties file for externalized strings. Use additional properties files, one per language

For details on configuring multilingual support, see the following topics:

- "Displaying Process Central in a Web Browsers Preferred Language " on page 462
- "Adding Multilingual Support for Forms and Tasks " on page 462

Displaying Process Central in a Web Browsers Preferred Language

If desired, you can display Process Central in the language of your choice. Each text string that displays in the Process Central application has been externalized to a file named `avoscentral.properties`.

This means you can provide a properties file with externalized strings for the wording of the login page, all command buttons, titles, error messages, and other wording.

`http://www.activevos.com` can provide the properties files for the languages you need. You can download a zip file and deploy the language BPRs to the Process Console, as provided in the `instructions.html` file in the zip archive. For details, go to the Software Development Kits (SDK) page of `http://www.activevos.com` and view the instructions in Process Central Language Pack.

For languages that `http://www.activevos.com` does not supply, you can provide the language values for externalized strings. The instructions for adding your own language file are included in the language pack archive.

To provide multilingual support for the Process Central user interface:

1. In a Web browser, go to `http://www.activevos.com/developers/sdks`.

2. Download Process Central Language Pack, Version 8 or later.

3. Unzip the file, and follow the instructions in `instructions.html` for modifying a language package.

See also "Naming Conventions for .properties File " on page 464.

Adding Multilingual Support for Forms and Tasks

You can create request forms as well as task forms (for a People activity) that support different languages. To do so, you create a new form, plus a default properties file and additional properties files for each language supported. The properties file contains externalized strings for schema-based keyIdentifiers. The steps to accomplish this support for requests are described below.

For details on task forms, see Providing Multilingual Support for Task Forms in the *Human Tasks* topics elsewhere in this help.

Create a new request form and a default properties file:

1. Create a new request form, described in "Creating a Process Central Process Form " on page 442.

2. In the Create Form dialog, select the **Advanced** button.

3. Select the Externalize String**s** check box.

4. Accept the default name and location for the properties file, or select the dialog button next to Property File to name and store your file. Do not use an underscore in the name.

When you externalize strings, the following events occur:

- Schema elements in the form are generated by identifier rather than by name to allow for runtime replacement of strings found in a properties file.

- A new link is added within the form to indicate multilingual support:

```
<link rel="i18n" charset="UTF-8" type="text/plain"
      href="myForm.properties" />
```

- The list of schema elements is added to the properties file as a key-value pair.

- The properties file that you create is the default externalized strings file.

For each additional language you want to support, do the following:

1. Make a copy of the default properties file and paste it into the forms folder.

2. Name the copied file with the syntax described in "Naming Conventions for .properties File " on page 464. Basically, you must use the exact same name for each file, followed by an underscore, followed by an ISO 639 Code for the Representation of Names of Languages.

3. Open the file in a text editor and replace the existing values with another language.

To test your properties, specify a language when you login to Process Central. For example:

```
http://localhost:8080/activevos-central/login.jsp?lang=fr
```

See also "I18N Functions " on page 249.

Adding Multilingual Support for an .avcconfig File

For an overview of this topic, see "Adding Multilingual Support to Process Central" on page 461.

You can provide externalized strings for the following attributes in an Process Central configuration file:

- Category names for the folders you add to Tasks, Forms, and Reports. For example, a Forms category named *Tutorial and Samples* contains the Process Developer tutorial request and the samples request forms.

- Descriptions for reports and forms. For example, the tutorial form contains a description that says, "Submit loan approval request (basic tutorial)."

- Names for the Task*Role* and *Show* picklists

To provide multilingual support:

1. Create a new `avcconfig` file, as described in "Creating a Process Central Configuration File " on page 449. Store it in your `deploy` folder. For example, `tutorial.avcconfig`.

2. For each name and description attribute in the file that you want to externalize, you must modify the place holder text to refer to a `${keyIdentifier}`. For example:

   ```
   <tns:requestCategoryDef id="education" name="${tutorial.and.samples}">
   ```

3. In a text editor, create a new `.properties` file that is the default externalized strings properties file for the `.avcconfig` file. For example, `tutorialConfig.properties`.

4. The `avcconfig` file must refer to this file, so you must add the following line, indicating multilingual support, in your `avcconfig` file:

   ```
   <tns:i18n location="/path/to/tutorialConfig.properties" />
   ```

5. In the default properties file, add each `${keyIdentifier}` and its English value. For example:
   `tutorial.and.samples=Tutorial and Samples`

6. As you add other externalized strings, add additional lines to the properties file, such as:

   ```
   tutorialRequest.description=Submit loan approval request (basic tutorial)
   ```

7. For each additional language, make a copy of the default properties file, and change the value of the keyIdentifier for that language.

   ```
   tutorial.and.samples=Tutoriel et échantillons
   ```

8. Name each file with an underscore and language name. For example:
   For example, `tutorialConfig_fr.properties`

9. Ensure the copy is UTF-8 encoded. By default Process Developer sets the encoding to ISO-8859-1.

After deployment, test your properties by specifying a language when you login to Process Central. For example:

```
http://localhost:8080/activevos-central/login.jsp?lang=fr
```

Example Configuration File Snippet:

```
<tns:requestCategoryDef id="education"
   name="${tutorial.and.samples}">
<avccom:description>
   ${tutorialRequest.description}
</avccom:description>
<tns:i18n location="/path/to/tutorialConfig.properties" />
```

Example default properties file;

```
tutorial.and.samples=Tutorial and Samples
tutorialRequest.description=Submit loan approval request (basic tutorial).
```

See also .

Adding Multilingual Support for Reports

Process Central displays reports created in the Business Intelligence Reporting Tools (BIRT) Designer. Within Process Central, the reports are displayed in the BIRT Viewer.

Process Central provides support for the following languages:

- German
- Spanish
- French
- Japanese
- Korean
- Simplified Chinese

If a browser's preferred language is set to one of the above, the BIRT Viewer automatically displays the report in that language.

To test for a language, provide a URL that specifies a language. For example:

```
http://localhost:8080/activevos-central/login.jsp?lang=zh_CN
```

To provide multilingual support for other languages, note the following considerations:

- To create properties files for the report design, refer to `http://www.birt-exchange.org/wiki/BIRT:Internationalization/`.
- To create properties for the BIRT Viewer text strings and error messages, for languages Process Central does not support, you must manually find and change several `messages.properties` files. The main file for the BIRT Viewer text strings is located in `server\webapps\activevos\WEB-INF\lib\ae_birtviewer.jar`.

Naming Conventions for .properties File

When you add multilingual support for Process Central configuration, forms and tasks, you create a properties file for each language you want to support. There is one default file and other files with language codes:

```
MyPropertiesFile.properties:
```

This is typically the default file for en-US.

```
MyPropertiesFile_ISO639Name.properties:
```

For each language, you must add an underscore and a ISO 639.2 code for the representation of names of languages. Add the _language code and optional country code to the default file name. Country codes are the upper-case, two-letter codes as defined by ISO-3166.

Examples:

- `requestFormOperation.properties`
- `requestFormOperation_fr.properties`
- `requestFormOperation_de.properties`
- `MyConfigFile.properties`
- `MyConfigFile_zh_CN.properties`

Process Central Advanced Configuration

In addition to providing configuration and multilingual support for Process Central forms, tasks, and reports, you can add more advanced configurations.

This topic covers the basics of how to implement additional configuration, including:

- Replacing the Process Central logo and icon images.
- Pointer to the Informatica Business Process Manager SDK and Documentation and Process Central Samples.

For advanced examples, check `http://www.activevos.com`.

To add your own logo:

1. In your project, create a text file with a name based on the default `avoscentral.properties` file. For example, `avoscentral_en.properties`.

2. Start the server, and open the Process Console.

3. Search for `avoscentral.properties`.
    This file contains keyIdentifiers and values for the Process Central user interface.

4. In the Resource Detail page of the Process Console, you will see the Resource Definition for the properties file.

5. To add your own logo to Process Central, copy the following line from `avoscentral.properties`:
    `ae.avc.header.logo.image=../avc-res/images/ActiveVOScentralTitle.png`

6. Paste this line into your properties file (`avoscentral_en.properties`).

7. Change the identifier value to specify the location and name of your logo. For example:
    `ae.avc.header.logo.image=project:/myProject/images/myLogo197x35.png`

8. Create a new `.bprd` file for the deployment of your logo and properties file.

9. In the `.bprd`, you see a catalog entry for your properties file. For example:
    ```
    <otherentry typeURI="res:/text/plain" location="project:/myProject/deploy/
    avoscentral_en.properties" classpath="resources/myProject/deploy/
    avoscentral_en.properties" />
    ```

10. Modify the location hint for your properties file. Your properties file must be stored in the catalog location provided by the location hint on the Resource Detail page of `avoscentral.properties`, as shown:

```
location="project:/com.activee.avoscentral.services/bundle/avoscentral_en.properties"
```

Tips:

- For a shortcut to deploying your file, you may prefer to create a project and export a file as described in the following topic: .

- In the Developers' How-To section of the Web site, see Process Central and Forms.

- Download the Informatica Business Process Manager SDK from the Product Documentation section of `http://www.activevos.com` for complete details of creating forms.

CHAPTER 34

Building a Process with a System Service

Select a template to begin building a system-service-based process.

Topics discussing building a process with a system service are:

- *Using a BPEL Template for a System Service-Based Process*
- *Alert Service*
- *Retry-Policy Service*
- *Identity Service*
- *Email Service*
- *Monitoring Alert Service*
- *Reporting Service*
- *Server Log Service*
- *Shell Command Invoke Service*
- *Data Access Service*
- *Migration Service*

Using a BPEL Template for a System Service-Based Process

System service-based processes interact with the Process Server in a variety of ways, such as sending an alert when a server event is triggered. The processes are based on interfaces that include both process consumer and partner service provider participants.

For process consumer interfaces, you can use the New BPEL Process wizard to start building your process. When you select **File > New> BPEL** and click next, you see the New BPEL Process from Template page.

This wizard page shows a list of processes you can create based on system-deployed WSDLs. You do not need to import any WSDL files to create a BPEL process. A process is started for you, with a receive activity, based on an operation from the WSDL resource.

The template generates a BPEL process based on one of the following WSDLs:

- Alert Process. For details, see *Alert Service*.

- Retry Policy Process. For details, see *Retry-Policy Service*.

- Custom B4P Notification. For details, see *Human Tasks* with the *Process Developer Help*.

- REST-based process. For details, see *Using a REST-based Service*.

- Event Action Process. For details, see *Creating an Event-Action BPEL Process*.

- Monitoring Alert. For details, see *Monitoring Alert Service*.

Note that when you deploy one of these processes, in the BPR, there are no WSDL or XSD. They are already deployed and available on the server.

In addition to the services listed above, there are other services you can build by starting in the Participants view. Using the System Services in the Partner Service Provider or Process Consumer wizard, you can use the operations for special purposes, including sending an email, or retrieving a user from an identity service. For details, see *System Services Interfaces*.

Alert Service

For many reasons an activity can throw an uncaught fault. Process Server provides the ability to suspend a process on an uncaught fault, and if desired, you can also trigger an alert.

When an uncaught fault occurs, the Process Server can instantiate an alert service that can then invoke some action, such as resuming the process by stepping over the faulting activity, automatically correcting a data value that is known to cause a fault, or simply notifying an administrator that a processing is faulting.

You can design a BPEL process to serve as the alert service. The process must be based on a WSDL file named `alert.wsdl`. There is a BPEL process template based on this WSDL, as described in the steps below. Also, this file is automatically loaded in Participants view as a System Service, so that you create a process consumer participant that adds a receive and reply for the process.

In essence, the WSDL defines the alert operation that receives information from the server regarding the process Id, namespace, name, variable location path, and status of the faulting process.

An input message sent to an alert-service BPEL process would have the following parts, shown in the example:

```
<ns1:alert xmlns:ns1=
    "http://www.active-endpoints.com/services/alert"
    processId= "101"
    processNamespace="http://tempuri.org"
    processName="myProcess"
    locationPath="/process/variables
        /variable[@name='testMessage']"
    status="suspended"
    faultName="someFault"
    faultNamespace="someFaultNamespace"
</ns1:alert>
```

Here are some general steps to consider when creating an alert-service BPEL process.

To create an alert-service BPEL process:

1. In the Project Explorer, create a new orchestration project.

2. Select **File > New > BPEL** Process.

3. Name the process and click **Next**.

4. On the Process Template page, select the Alert process template.

A new file opens on the Process Editor canvas with a *ReceiveAlert* activity. The receive activity is based on the alert operation.

5. Finish building the process, adding the programming logic that you wish to occur for your use case when a process is faulting. For example, send an email, described in "Email Service " on page 479.

6. Launch the Deployment Descriptor wizard and on the Partner Links tab, do the following:

    a. Select the AlertPL My Role partner link.

    b. Type in a *Service Name* for the My Role partner link.

7. Start up the Process Server and deploy the alert-service process to the server. Note that in the BPR, there are no WSDL or XSD. They are already deployed and available on the server.

8. In the Process Console of the server, add the service to the Alert Service page. See the *Process Console Help* for details.

Tip: Within `http://wwww.activevos.com`, refer to *Handling Process Server Alerts using Email and Identity Services* for a comprehensive sample alert service that you can download.

Data Access Service

You can interact directly with a data source within a BPEL process by using the Data Access system service. Using this service, you create an invoke activity that executes one or more SQL statements on a specified data source and receives a result set as the response.

The Data Access service provides two operations: `execSQL` and `execMultiSQL`. These operations allow you to execute either a single SQL statement or multiple SQL statements. You can execute any SQL statement, whether you are selecting, inserting, updating or deleting data or stored procedures from a data source.

During deployment, you must specify the target data source.

Creating a BPEL Process that Executes Statements on a Data Source

Use the following to create a BPEL process that executes statements on a data source:

In the Project Explorer, create a new orchestration project.

1. Select F**ile > New > BPEL Process**.

2. Name the process and click **Finish**.

3. In the Participants view, create a new Partner Service Provider.

4. Select System Service**s** from the Interfaces tree.

5. Select *Data Access*.

6. From the Partner Service Provider that you created, drag either the `execSQL` or `execMultiSQL` operation to the canvas to create an invoke activity.

7. Fill in the input data, with the selections shown in the examples below the table.

The following table shows the data access request message element.

Following this table are the following examples:

*   Example request for a single statement and its response

*   Example request including multiple statements and its response

- Parameter-based request and responses from an insert, update, or delete statement
- Handling binary data
- An example fault response

Examples of invoking stored procedures; examples are shown for MySQL, SQL Server, and Oracle.

execSQL Input Element	Description
sqlStatement	Element containing the required `<statement>` and optional attributes
includeMetadata	Returns a row of metadata about the query columns prior to the actual return data (the default is false). Meta data includes data type and display size.
maxRows	Maximum number of result rows to be returned. (the default is 0 or unlimited)
maxWaitSeconds	Maximum time to wait for results. (The default is 0 or unlimited.)
columnCase	Specifies the case formatting of the columns in the result set (the default is *unchanged* from server). Schema enumerations defined for this element include lowercase, uppercase, and unchanged.
statementId	Specified as part of request to help identify results (the default is to auto generate an Id that results in an Id such as statement-1, statement-2)
statement	(Required.) SQL statement to execute
parameter	Identifies the data to be inserted, updated or deleted from the data source. Parameter values are listed in order within a `<parameterBatch>` element associated with a statement.
sqlType	Parameter attribute. Types include string (default), byte, short, int, long, float, double, date, binary, or clob
attachmentId	Parameter attribute. Specified as part of request to help identify results attachments, if any. S
hasResultSet	Indicates that a stored procedures returns data. Only use this parameter if this is true.

Example request for a single statement:

```
<das:dataAccessRequest xmlns:das="http://schemas.active-endpoints.com/data-access/
2010/04/data-access.xsd">
   <das:sqlStatement includeMetadata="true" maxRows="100" maxWaitSeconds="120"
columnCase="lowercase">
   <das:statement>SELECT EngineId, Name, State FROM AeEngine ORDER BY 1</das:statement>
   </das:sqlStatement>
</das:dataAccessRequest>
```

Example response from the above request:

```
<dataAccessResponse xmlns="http://schemas.active-endpoints.com/data-access/2010/04/data-
access.xsd"
                    statementId="statement-1" >
   <metadata>
      <engineid dataType="INT UNSIGNED" displaySize="10" xmlns=""/>
      <name dataType="VARCHAR" displaySize="255" xmlns=""/>
      <state dataType="TINYINT" displaySize="4" xmlns=""/>
   </metadata>
   <row>
      <engineid xmlns="">1</engineid>
      <name xmlns="">machine1</name>
```

```
        <state xmlns="">1</state>
    </row>
</dataAccessResponse>
```

Example request including multiple statements:

```
<das:multiDataAccessRequest xmlns:das="http://schemas.active-endpoints.com/data-access/
2010/04/data-access.xsd">
  <das:sqlStatement includeMetadata="false" statementId="num-processes">
    <das:statement>SELECT COUNT(*) AS ProcessCount FROM AeProcess</das:statement>
    </das:sqlStatement>
  <das:sqlStatement includeMetadata="false" statementId="num-plans" sqlcode='0'
sqlstate='0'>
    <das:statement>SELECT COUNT(*) AS PlanCount FROM AePlan</das:statement>
  </das:sqlStatement>
  <das:sqlStatement includeMetadata="false" statementId="metainfo">
    <das:statement>SELECT * FROM AeMetaInfo</das:statement>
  </das:sqlStatement>
</das:multiDataAccessRequest>
```

Example response from the above multiple statement request:

```
<multiDataAccessResponse xmlns="http://schemas.active-endpoints.com/data-access/2010/04/
data-access.xsd">
  <result statementId="num-processes">
    <row>
        <ProcessCount xmlns="">3</ProcessCount>
    </row>
  </result>
  <result statementId="num-plans">
    <row>
        <PlanCount xmlns="">18</PlanCount>
    </row>
  </result>
  <result statementId="metainfo">
    <row>
        <PropertyName xmlns="">DatabaseType</PropertyName>
        <PropertyValue xmlns="">mysql</PropertyValue>
    </row>
    <row>
        <PropertyName xmlns="">SiteId</PropertyName>
        <PropertyValue xmlns="">0</PropertyValue>
    </row>
    <row>
        <PropertyName xmlns="">SiteName</PropertyName>
        <PropertyValue xmlns="">mySite1</PropertyValue>
    </row>
    <row>
        <PropertyName xmlns="">Version</PropertyName>
        <PropertyValue xmlns="">1.2.3 ActiveVOS Enterprise</PropertyValue>
    </row>
  </result>
</multiDataAccessResponse>
```

Parameter-Based Request

You can create a SQL statement that modifies data rather than returns data. The statement can take the form of a SQL INSERT, UPDATE or DELETE operation. A set of parameters for the operation describe the data to modify. In the following example, an insert statement adds a property name and value to each row of a table. The properties are specified in the statement, and the property values are specified as parameters:

```
<das:dataAccessRequest
        xmlns:das="http://schemas.active-endpoints.com/data-access/2010/04/data-
access.xsd">
  <das:sqlStatement>
    <das:statement>insert into myTable(PropertyName,
 PropertyValue) VALUES (?, ?)</das:statement>
    <das:parameterBatch>
        <das:parameter sqlType="string">Property1</das:parameter>
```

```
        <das:parameter sqlType="string">One</das:parameter>
    </das:parameterBatch>
    <das:parameterBatch>
        <das:parameter sqlType="string">Property2</das:parameter>
        <das:parameter sqlType="string">Two</das:parameter>
    </das:parameterBatch>
    <das:parameterBatch>
        <das:parameter sqlType="string">Property3</das:parameter>
        <das:parameter sqlType="string">Three</das:parameter>
    </das:parameterBatch>
  </das:sqlStatement>
</das:dataAccessRequest>
```

Simulating the Data Access Invoke Output

You can test the actual Data Access request and response during simulation by performing some additional steps prior to beginning simulation. The steps involve using the Eclipse Data Source Explorer that is bundled with Process Developer.

Using the Data Source Explorer, you can generate sample data from an actual statement execution and export the data to Process Developer in the correct schema format. This technique makes the actual data returned from the result set available to a process variable to which you are mapping data.

Step 1. Set up a database connection:

1. Select **Window > Show View > Other > Data Management > Data Source Explorer**.

2. Right-mouse click on Database Connections and select **New**.

3. Select a Connection Profile Type from the list. For example, select MySQL.

4. Name the new profile, and select **Next**.

5. On the Driver page, select your database driver from the list. If the driver is not in the list, select **New Driver** and specify a driver definition.

6. Enter the URL to connect to your database. For example, the default URL for MySQL is `jdbc:mysql://localhost:3306/[database name]`.

7. Select Test the Connection, and if successful, finish the wizard.

8. Notice there is a new database listed in the Data Source Explorer.

Step 2. Connect to the database and execute a statement:

1. In the Data Source Explorer, right-mouse click on your database and select Connect if needed. Expand the tree, and you can see an outline of all the tables in your database.

2. Right-mouse-click on your database and select Open SQL Scrapbook.

3. Select your database from the Name list.

4. Your Data Source Explorer and SQL Scrapbook window should look similar to the following example:
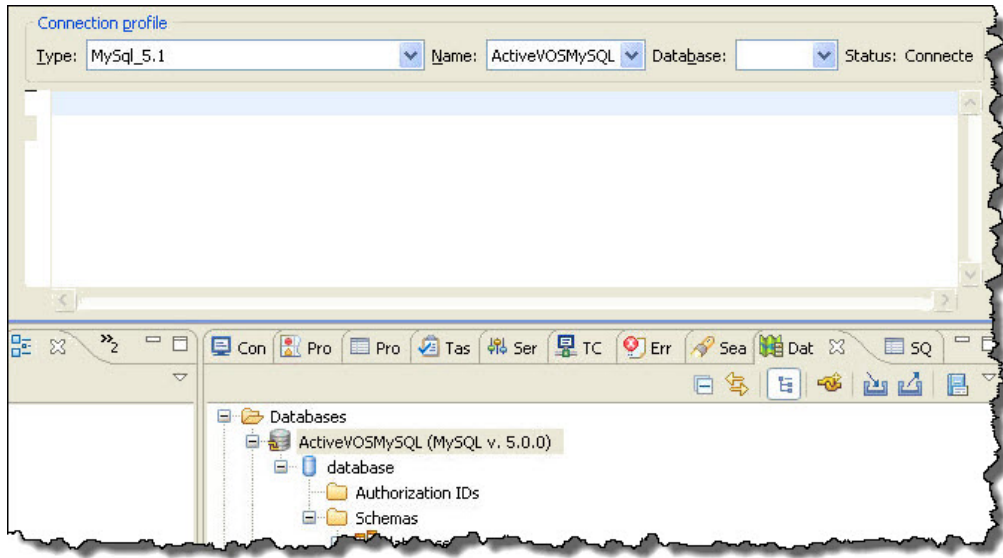
5. If your Data Access invoke is based on the `execSQL` operation, copy and paste your single execution statement into the SQL Scrapbook.

6. Right-mouse click and select **Execute Selected Text**. Notice that the results are returned in the SQL Results tab.

7. For the `execMultiSQL` operation, copy and paste your statements into the SQL Scrapbook. Right-mouse click and select **Execute All**.

Step 3. Save the SQL Results as sample data:

1. In the SQL Results view, right-mouse click and select **Save > Current Result**.
   You must select Current Result for a a single statement execution and All Results for multiple statement execution. The format of the data differs depending on your selection.
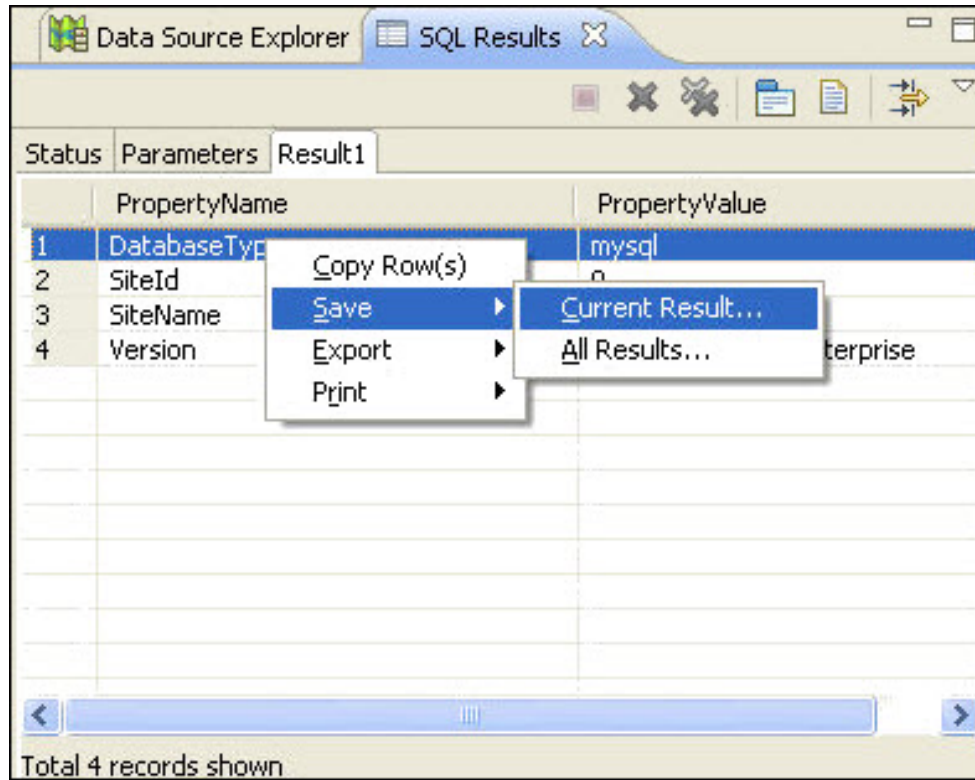
2. In the **Save Result** dialog, browse to your Data Access project in your workspace and select the `sample-data` folder. Name and save your file into this location.

3. In the File Type box, select Process Developer Data Access Sample. Then set the Output encoding to UTF-8. The following example shows all settings:
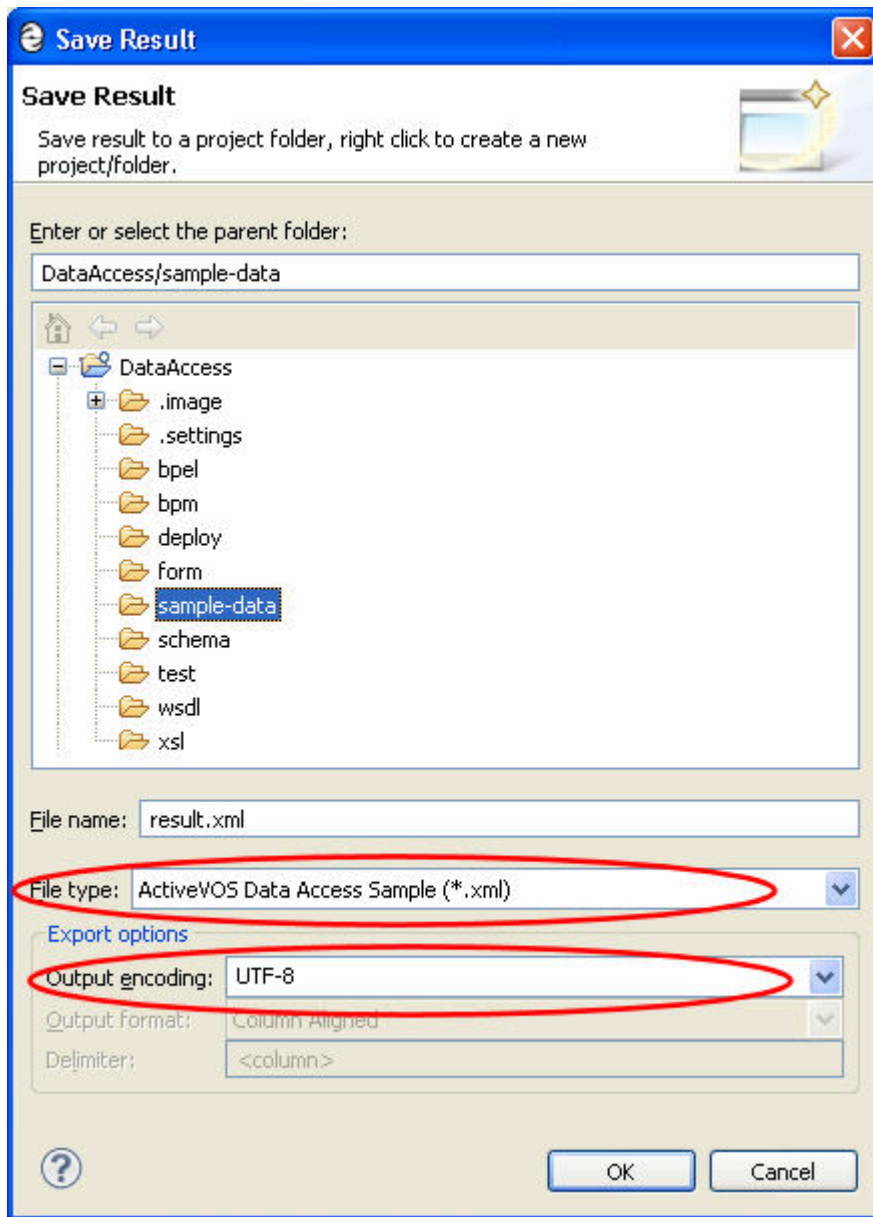
4. Select **OK**.

Example-Response from an Insert Update or Delete Statement

No data is returned from a non-query statement. The response contains only the number of rows affected by the operation, as the example shows:

```
<dataAccessResponse
    xmlns="http://schemas.active-endpoints.com/data-access/
        2010/04/data-access.xsd"
    statementId="statement-1">
  <row updatedRows="3"/>
</dataAccessResponse>
```

Handling Binary Data

You can handle an image, document and other binary data as an attachment or a base64-encoded string. You can include the binary data in the form of a message attachment, and then specify the `attachmentId` as part of the `SqlParameter` in the insert specification. For details on attaching binary data to a message, see [Attachments](). You can include base64-encoded strings within the request as the following examples show.

Example One: Insert a base64-encoded string

```
<das:sqlStatement statementId="insert-blob" columnCase="lowercase">
      <das:statement>INSERT INTO myTable (SmallNumber, SampleBlob) VALUES (?, ?)</
das:statement>
      <das:parameterBatch>
          <das:parameter sqlType="short">11</das:parameter>
          <das:parameter sqlType="binary">iVBO.../Ay...+
                        fPmt..(incomplete)</das:parameter>
      </das:parameterBatch>
   </das:sqlStatement>
```

Example Two: Select binary data

```
<das:sqlStatement statementId="get-blob" columnCase="lowercase">
      <das:statement>SELECT SmallNumber, SampleBlob
                FROM AeMyTable
                WHERE SmallNumber = 11</das:statement>
   </das:sqlStatement>
```

Fault Handling

The Data Access operations include a fault message that returns a fault for an SQL exception. The data returned includes an error message, `sqlcode` and `sqlstate`. All other faults are system faults.

Example fault response

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Body>
      <env:Fault>
         <env:Code>
            <env:Value>env:Receiver</env:Value>
            <env:Subcode>
               <env:Value xmlns:ns1=
                  "http://docs.active-endpoints/wsdl/
                     data-access/2010/04/
                     data-access.wsdl">ns1:dataAccessFaultMessage
               </env:Value>
            </env:Subcode>
         </env:Code>
         <env:Reason>
            <env:Text xml:lang="en-US">Table
               'activevos.aeprocessx' doesn't exist</env:Text>
         </env:Reason>
         <env:Detail>
            <das:dataAccessFaultMessage sqlcode="1146"
               sqlstate="42S02"
               xmlns:das="http://schemas.active-endpoints.com/
                        data-access/2010/04/data-access.xsd">
            <errorMessage xmlns="http://schemas.active-endpoints.com/
                        data-access/2010/04/data-access.xsd">Table
                              'activevos.aeprocessx' doesn't exist</errorMessage>
            </das:dataAccessFaultMessage>
         </env:Detail>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

Examples

Example of running a stored procedure on a MySQL database

```
<das:multiDataAccessRequest
    xmlns:das=
        "http://schemas.active-endpoints.com/data-access
            /2010/04/data-access.xsd">
    <das:sqlStatement columnCase="uppercase"
            statementId="create-proc">
        <das:statement>CREATE PROCEDURE AeTestCount(OUT result INT)
                    BEGIN
                        SELECT COUNT(*) INTO result
                                    FROM AeJUnitTestTable;
                    END
        </das:statement>
    </das:sqlStatement>
    <das:sqlStatement columnCase="uppercase"  statementId="call-proc">
        <das:statement>CALL AeTestCount(@rowCount);</das:statement>
    </das:sqlStatement>
    <das:sqlStatement columnCase="uppercase"
                    statementId="get-results">
        <das:statement>SELECT @rowCount;</das:statement>
    </das:sqlStatement>
    <das:sqlStatement columnCase="uppercase"
                        statementId="drop-proc">
        <das:statement>DROP PROCEDURE AeTestCount</das:statement>
    </das:sqlStatement>
</das:multiDataAccessRequest>
```

Example of running a stored procedure on an SQL Server database

```
<das:multiDataAccessRequest xmlns:das=
        "http://schemas.active-endpoints.com/
                    data-access/2010/04/data-access.xsd">
    <das:sqlStatement columnCase="uppercase" statementId="create-proc">
        <das:statement>CREATE PROCEDURE    AeTestCount @num_rows INT OUTPUT
                    AS
                    BEGIN
                        SELECT @num_rows = COUNT(*)
                                    FROM AeJUnitTestTable
                    END
        </das:statement>
    </das:sqlStatement>
    <das:sqlStatement hasResultSet="true" columnCase="uppercase"
                            statementId="call-proc">
        <das:statement>DECLARE @result INT
                    EXEC AeTestCount @num_rows = @result OUTPUT
                    SELECT @result RESULT
        </das:statement>
    </das:sqlStatement>
    <das:sqlStatement columnCase="uppercase"
                                statementId="drop-proc">
        <das:statement>DROP PROCEDURE AeTestCount</das:statement>
    </das:sqlStatement>
</das:multiDataAccessRequest>
```

Example of running a stored procedure on an Oracle database

```
<das:multiDataAccessRequest
 xmlns:das="http://schemas.active-endpoints.com/
                data-access/2010/04/data-access.xsd">
    <das:sqlStatement columnCase="uppercase"
                    statementId="create-proc">
        <das:statement>create or replace
                        PROCEDURE AeTestCountProc
                            (PROP_NAME IN VARCHAR2, PROP_VAL IN VARCHAR2,
                                TOTAL OUT NUMBER)
                        AS
                            BEGIN
                                UPDATE AeMetaInfo SET PropertyValue = PROP_VAL
                                    WHERE PropertyName = PROP_NAME;
```

```
                                SELECT COUNT(*) INTO TOTAL
                                FROM AeJUnitTestTable;
                        END;
    </das:statement>
  </das:sqlStatement>
  <das:sqlStatement columnCase="uppercase"
                    statementId="call-proc">
    <das:statement>{CALL AeTestCountProc(?,?,?)}</das:statement>
    <das:procedure>
        <das:parameter name="PROP_NAME"
                    sqlType="string" mode="in">BogusName</das:parameter>
        <das:parameter name="PROP_VAL" sqlType="string"
                        mode="in">BogusValue</das:parameter>
        <das:parameter name="RESULT" sqlType="int" mode="out" />
    </das:procedure>
  </das:sqlStatement>
  <das:sqlStatement columnCase="uppercase"
                    statementId="drop-proc">
    <das:statement>DROP PROCEDURE AeTestCountProc</das:statement>
  </das:sqlStatement>
</das:multiDataAccessRequest>
```

Mapping Data Returned in the Data Access Response

You can map data from a response by writing an XPath expression that selects nodes from the rows returned. The XPath expression is used in a copy operation in an assign activity or directly in the Output data tab of the data access invoke activity.

For example, suppose the request message contains a statement such as:

```
SELECT EngineId, Name, State FROM AeEngine ORDER BY 1
```

You can extract a value, such as the State value in the first row of the `AeEngine` table with the following XPath expression:

```
$dataAccessResponse//ns1:row[0]/state/text()
```

Simulating the Data Access Service

When you create the Process Deployment Descriptor for the process containing the Data Access service, you must specify the location of your database. Complete the deployment information as follows:

1. On the Partner Links page, notice that the Partner Service Provider's Invoke Handler for data access is automatically set to System Service.

2. If you are using a cloud process, in the edit field, replace the text, FILL_IN_DATASOURCE_JNDI, with one of the following:

    *   The JNDI Name defined in your application server that is bound to the data source. There is no need to include the JNDI prefix. For example:

        ```
        jdbc/myDatabase
        ```

    *   Similarly, for WebLogic and Websphere, no prefix is required:

        ```
        jdbc:/myDatabase
        ```

    *   A URN that can be mapped to a JNDI Name or URL in the Process Console URN Mappings page. This option gives you a way to change the database connection information without refactoring and redeploying your process.

Note : If you are deploying on-premise, you must also include the JNDI prefix as shown in this example:

```
java:comp/env/jdbc/myDatabase
```

For preliminary testing, if you want to set up a database connection within the server embedded in Process Developer, follow these instructions:

1. In Process Developer, stop the embedded server, if it is running. For details, see Setting Up the Embedded Process Server.

2. In the file system, in your Process Developer installation directory, navigate to the embedded server plugin. For example, [*designer install directory*]`\designer\plugins`
   `\org.activebpel.enginep_[`*versionNNN*`]` where versionNNN is the plugin version.

3. In the `org.activebpel.enginep` plug-in, navigate to `server\conf\Catalina\localhost` and make a backup copy of the `active-bpel.xml` file. Then open the file.

4. In the `active-bpel.xml` file, add a new `<Resource>` element within the `<Context>` element, as shown in the example:]

```
<Resource
    driverClassName="driver_class_name"
    maxActive="100"
    maxIdle="30"
    maxWait="10000"
    name="jdbc/anyName"
    password="password"
    type="javax.sql.DataSource"
    url="url_of_your_db"
    username="db_login_name"/>
```

Be sure to replace all attribute values with values for connecting to your database.

5. Save the file and restart the embedded server.

6. Be sure to add the JNDI name to the PDD of your process before deploying your BPR to the embedded server.

Email Service

Process Server supports an email service. This means you can create a process that sends email messages to a list of recipients via an invoke activity. Many types of processes can benefit from an email delivery activity, including use an "Alert Service " on page 468.

Any BPEL process that implements email-based activities imports the WSDL provided with the ProcessServer. The name of the WSDL is `email.wsdl`, and in the Participants view wizards, it is a System Service named *send email*. The WSDL contains a send operation that can be used in an invoke activity as follows:

```
<invoke
    inputVariable="emailMessage"
    operation="send"
    outputVariable="resultMessage"
    partnerLink="emailProvider"/>
```

To create a BPEL process that sends an email:

1. In the Project Explorer, create a new orchestration project.

2. Select **File > New > BPEL Process**.

3. Name the process and click Finish.

4. In the Participants view, create a new Partner Service Provider.

5. Select System Services.

6. Select *Email*.

7. From Participants view, drag the **send** operation to the canvas to create an invoke activity. This activity sends the email message to people in your identity service enabled on the Process Server.

8. Add additional programming logic. For example:

    a. Add a receive activity that receives the input to be associated with the email.

    b. Use the input tab of the invoke to assign data to the variable. See the examples below.

9. Use the send operation's fault message to catch communication errors with your email service.

Example to initialize the emailMessage variable:

1. In the **Input** tab of the invoke, select XPaths as the Assignment Type.

2. Select **Add**.

3. In the **E/L** panel, select Literal.

4. In the From column, select the dialog button and generate literal contents. Process Developer automatically finds the correct schema for the selected single-part element-based variable and generates a literal template for you as shown:
   Generated Template

```
<aem:emailMessage xmlns:aem="http://schemas.active-endpoints.com/email/2007/01/
email.xsd">
    <aem:from>string</aem:from>
    <aem:replyTo>string</aem:replyTo>
    <aem:to>string</aem:to>
    <aem:cc>string</aem:cc>
    <aem:bcc>string</aem:bcc>
    <aem:subject>string</aem:subject>
    <aem:body mimeType="string"></aem:body>
</aem:emailMessage>
```

To avoid errors, be sure to set the body mimeType to a valid type such as *text/plain*. See the example in Step 5.

5. Use one of the following tips to populate the `emailMessage` elements with the appropriate strings:

   - Create XPaths expressions, one per element. For example, to populate the <aem:to> element, create an expression such as FROM`$quoteRequest/quote:contact/quote:name()` TO `aem:to`.

   - Create an expression with a `doXslTransform` function. This solution requires that you create an XSL file containing all the transformation details for the email elements. For example, `doXslTransform ('monitorEmail.xsl', $handleAlertRequest)`

   - Set the Assignment Type to XQuery. Then you can add {expressions} or strings directly to each `emailMessage` element.

Example

```
<ns:emailMessage
    xmlns:ns="http://schemas.active-endpoints.com/email/2007/01/email.xsd">
  <ns:from>Dave Verd &lt; DVerd@mycompany.com &gt;</ns:from>
  <ns:replyTo>sender@mycompany.org</ns:replyTo>
  <ns:to>{$quoteRequest/quote:contact/quote:name()}</ns:to>
  <ns:subject>{ $quoteRequest/quote:contact/quote:description}</ns:subject>
  <ns:body mimeType="plain/text">Message goes here.</ns:body>
</ns:emailMessage>
```

Deploying a Process as an Email Service

You can deploy your process and make it available as a standard document literal service for the My Role partner link. Note that in the BPR, there are no WSDL or XSD. They are already deployed and available on the server.

If your email list contains an invalid address, the process logs an error. Check the Server Log in the Process Console to view email errors.

Identity Service

An identity service provides a way to look up users and groups based on a defined set of attributes. The Process Server support for an identity service is based on Lightweight Directory Access Protocol (LDAP), JDBC, or a file-based service.

You can create a process that includes identity-based activities. As a prerequisite, in the Process Console, you must provide the communication details for access to your directory service. When the process runs, the user or group specified in the process is looked up in your directory service.

Any BPEL process that implements identity-based activities imports the WSDL provided with the Process Server. The name of the WSDL is `identity.wsdl`, and in the Participants view wizards, it is a System Service named *identity search*.

The WSDL contains the following operations:

Operation Name	Description
findRolesByPrincipal	Returns a list of roles for the named principal. For example *User1* is a member of *Administration* and *Finance*.
findRoles	Returns a list of roles, such as *Marketing*, *Finance*, and *DnsAdmins* that are defined in the directory service.
findIdentitiesByRole	Returns a list of identities for the named roles
findIdentities	Returns a list of identities that includes user name and email address Specifies which roles and principals to include and exclude from the results The query is comprised of include and exclude statements. The roles or principals listed under the include element are included in the result set while the ones listed under the exclude element are excluded.
assertPrincipalInQueryResult	For a given principal, make sure the user exists in the final result of the query. You can use this operation for fault handling. It is good for checking permissions.
assertPrincipalInQueryResultwithResponse	Same as above with response
countIdentities	Receive a count of the users and groups requested

Building an Identity-Based Activity

Use identities in an alert service, for example, to send email to a group when a process is suspended on an uncaught fault.

You can also use identities in a BPEL process for branching, based on a group. For example, build one branch of an activity for managers, another for customer service representatives.

Build the assign activities you need to invoke the identity service. You can generate literal contents for a variable based on the `aeid:identityQuery` element.

Here is an example:

```
<aeid:IdentityQuery xmlns:aeid="http://schemas.active-endpoints.com/identity/2007/01/
identity.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="aeid:IdentityQuery">
<aeid:include>
```

```
<aeid:group>Development</aeid:group>
<aeid:user>user1</aeid:user>
<aeid:id>CN=Kim Pan,CN=Users,DC=aedomain,DC=active-endpoints,DC=local</aeid:id>
</aeid:include>
</aeid:IdentityQuery>
```

Note that the `<aeid:id>` element above shows an example of looking up a user in a LDAP directory by distinguished name (DN). For JDBC, the lookup is the primary key in the database. For `tomcat-users.xml`, the id is the same as the user name.

Add other programming logic, as desired.

In the PDD, for the partner role, System Service is pre-selected as a custom invoke handler. The PDD entry looks similar to the following:

```
<partnerLink name="provider">
    <partnerRole endpointReference="dynamic"
            invokeHandler="system"/>
</partnerLink>
```

Deploying a Process as an Identity Service

You can deploy your process and make it available as a standard document literal service for the My Role partner link. Note that in the BPR, there are no WSDL or XSD. They are already deployed and available on the server.

Migration Service

The migration service allows fine grain control over migrating running process instances to a newer (or different) version of a process definition. The simplest way to migrate running processes is to specify this choice in the Process Deployment Descriptor of the newer version being deployed, on the "General Deployment Options " on page 348 page.

However, if the newer process version has significant changes, such as new activities, variables, or participants, you can create a migration service to test the migration, by selecting one or a few process instances to migrate. Then analyze the results on the Active Process Details page of the server. If errors occur, a process is suspended and errors are reported in the server log. Please note that Process Server does not validate changes to incoming and outgoing message structures between the existing plan and the new plan.

Another use case for a migration service is to have fine grain control over migration. For example, you don't have to migrate instances to the online version of the same process definition. You can migrate to any other process definition that is on the server. And you can pinpoint only those running process instances that you want to migrate.

The process you create for handling migration uses the Migrate system service that has two operations that you must use in order:

- **Create Map operation**
  This operation takes as input the Plan Id of the process definition that process instances are running against. Each Plan Id on the server is unique and is a property of each version of a process definition. All process instances run against a Plan Id.

  The create map operation returns a migration map XML document as a string. The map contains the migration structure between plans, which describes the transformation of the activities, links, and variables from one process definition to the other.

- **Migrate operation**

  This operation takes as input the migration map and the process id (pid) of the running process instance to migrate.

  The migrate operation returns the XML document that contain list of migration warnings (if any) and the backup process Id. By default, the process instance is migrated to the Plan Id of the online version of the same process definition, but you can specify any Plan Id on the server.

Input and Output for Create Map

The input for the create map operation looks like the following example. You must provide a Plan Id for the running process instances to migrate. For your process request message that passes data to this operation, you can look up the Plan Id on the Active Process Detail page or obtain it programmatically in various ways, such as a using a database query or the appropriate Admin API call.

The following example show sample data. Note that the `toPlanId` is optional. If deleted, the default behavior is to map to the online version of the same process definition.

```
<migration:createMapRequest xmlns:migration="http://docs.active-endpoints/wsdl/
migration/2011/04/migration.xsd">
    <migration:fromPlanId>144</migration:fromPlanId>
    <!--Optional. If not used, defaults to the online plan of the pid:-->
    <migration:toPlanId>145</migration:toPlanId>
</migration:createMapRequest>
```

The following example shows just the beginning of the migration map created for the response:

```
<createMapResponse xmlns="http://docs.active-endpoints/wsdl/migration/2011/04/
migration.xsd">
    <migrationMap><![CDATA[
        <processMigration xmlns="http://schemas.activevos.com/processMigration/2010/05/
processMigrationMapper.xsd">
            <migrationInfo>
                <locationInfo>
                    <locationId>1</locationId>
                    <locationPath>/process</locationPath>
                    <idsToRoot />
                </locationInfo>
                <mapForVariable>false</mapForVariable>
                <migrationMapper>
                    <oldLocationInfo>
                        <locationId>1</locationId>
                        <locationPath>/process</locationPath>
                        <idsToRoot />
                    </oldLocationInfo>
                    <activityStrategy>
                        <stateMapType>0</stateMapType>
                        <defaultLocationInfo>
                            <locationId>1</locationId>
                            <locationPath>/process</locationPath>
                            <idsToRoot />
                        </defaultLocationInfo>
                    </activityStrategy>
                </migrationMapper>
            </migrationInfo>
...
</createMapResponse>
```

Input and Output for Migrate

The input for the migrate operation requires a pid. For your process request message that passes data to this operation, you can look up the pid on the Active Processes list or obtain it programmatically. Basically you can use the `getProcessList` API call (or a report) to bring back the set of process ids matching the ones you want to migrate. If you have a large number to migrate, you can migrate many instances in parallel.

The following example show sample data:

```
<migration:migrateRequest xmlns:migration="http://docs.active-endpoints/wsdl/migration/
2011/04/migration.xsd">
<!--The string comes from the Create Map operation: -->
    <migration:migrationMap>string</migration:migrationMap>
    <migration:pid>850</migration:pid>
    <!--Optional. If not used, defaults to the online plan of the pid. If used, must
match with toPlan in migrate operation:-->
    <migration:toPlanId>145</migration:toPlanId>
    <!--Optional. Defaults to false:-->
    <migration:createBackup>false</migration:createBackup>
</migration:migrateRequest>
```

The following is a sample migrate response message:

```
 <migrateResponse xmlns="http://docs.active-endpoints/wsdl/migration/2011/04/
migration.xsd">
        <response>MIGRATION_WARNING Errorcode:3001. Migration Warning: The empty
activity is being marked as Finished when it is not. This might result in invalid output
for the pid 850. (location path /process/sequence/empty)</response>
        <backupPid>851</backupPid>
    </migrateResponse>
```

Creating a BPEL Process That Migrates Process Instances to a New Version

Use the following procedure:

1. In the Project Explorer, create a new orchestration project.

2. Select **File > New > BPEL Process**.

3. Name the process and click **Finish**.

4. Create a receive/reply based on a WSDL you create. The request message for the process must provide a Plan Id and one or more process Ids of running process instances.

5. In the Participants view, create a new Partner Service Provider.

6. Select System Services.

7. Select *Migration*.

8. From Participants view, drag the **create map** operation to the canvas to create an invoke activity. This activity takes as input the process plan Id and creates a map of the process structure.

9. From the Participants view, drag the **migrate** operation below the create map invoke to create a second invoke activity. This activity takes as input the migration map and process id and migrates the running process instance to the online version of the same plan Id, by default. You can specify any Plan Id, depending on the structure of your process.

Testing Your Migration Process

Deploy the migration process to the server and then you can test it as follows:

1. Deploy a process and instantiate a long-running process instance.

2. Deploy a second version of the process. Do not select the migrate option in the PDD.

3. Instantiate the migration process with a request message containing a Plan Id of the original version and the pid of the running process instance.

For doing migration programmatically outside of a BPEL process, refer to "Using the Process Server Migration Web Service " on page 485.

Using the Process Server Migration Web Service

For doing migration programmatically outside of a BPEL process, you can use the Process Server SOAP-based migration Web service.

On the server, open the following page:

`http://localhost:8080/active-bpel/services`

On this page, scroll down to locate the `AeProcessMigrationService`:

`h``ttp://localhost:8080/active-bpel/services/AeProcessMigrationService?wsdl`

Using a tool such as SOAP UI, create a request to migrate one process instance, as shown in the following example:
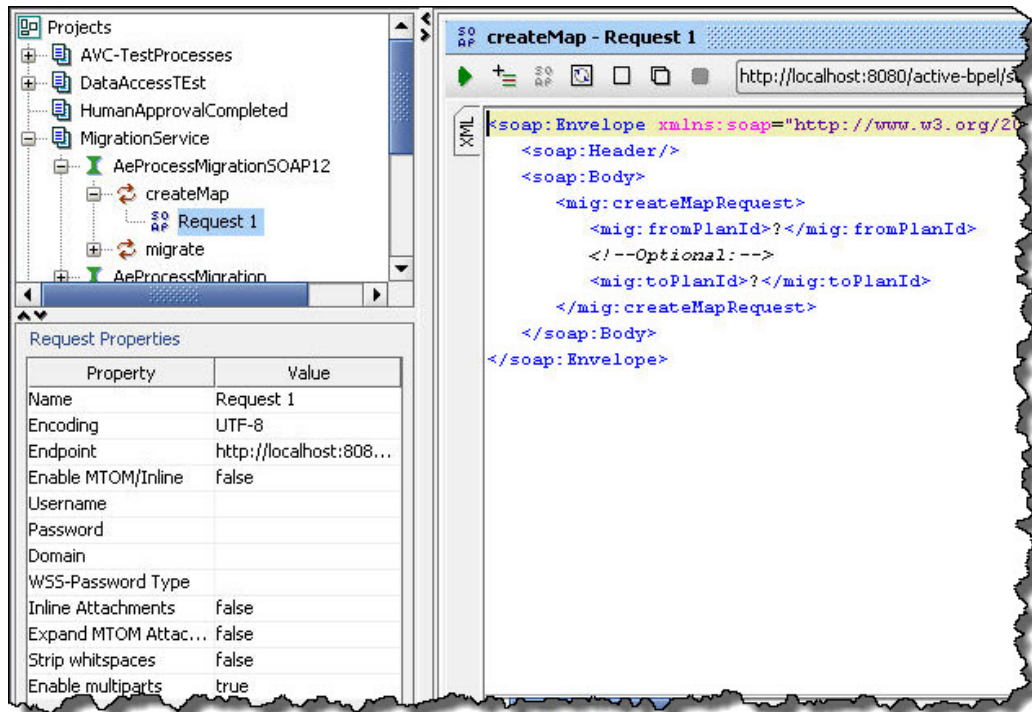
Monitoring Alert Service

Process Server provides engine monitoring, and if desired, you can trigger an alert to notify an administrator when an engine stops running or when a monitored property has a warning or error condition.

When you configure engine monitoring properties in the Process Console, the Process Server can instantiate a monitoring alert service. This service receives an alert from the engine, and the process can then notify an administrator about error conditions via email. For details on sending an email, see "Email Service " on page 479.

You can design a BPEL process to serve as the monitoring alert service. The process must be based on a WSDL file named `monitorAlert.wsdl`.

In essence, the WSDL defines the `handleAlert` operation that receives information from the server regarding engine status and engine properties being monitored, their monitoring level, threshold and value.

Retry-Policy Service

At times an endpoint may not reply when it is invoked in a BPEL process. In normal circumstances, the invoke faults. To avoid this, you can attach a policy to the endpoint reference in the PDD file to indicate that additional retry attempts are allowed at a specified interval for the endpoint. This technique is described in *Process Developer > Part IV: Testing and Deployment > Process Deployment > Endpoint Reference Addressing Considerations*.

Alternately, you can name a service in the policy section of the PDD file whose job it is to provide a retry interval, that you may want to calculate dynamically, and do additional work programmatically. The service could also specify an alternate endpoint to try when the invoke is faulting or suspended.

This service, which is a retry-policy service, is a process you can create and that the Process Server instantiates when an invoke activity is faulting (or suspended). The retry-policy process must be based on a WSDL named `retryCheck.wsdl`.

In essence, the WSDL defines the `retryCheck` operation that receives information from the server regarding the fault name, process Id, and other details of the faulting invoke activity. The process you write is instantiated when the invoke activity faults (or can be suspended), and it receives an input message identifying details of the fault.

Server Log Service

Process Server provides a Server Log that captures the details of server events, such as configuration changes and errors in running processes. You can create a process to interact with the Server Log so that you can add your own message to the log programmatically. This interaction provides a way to troubleshoot your running processes.

The Server Log page is listed under the Monitor menu of Process Console.

The input for the `logMessage` looks like the following example:

```
<ns:logWriteMessage xmlns:ns="http://schemas.active-endpoints.com/logging/2009/05/
logging.xsd">
   <ns:message>My message to insert into server log
   </ns:message>
    <ns1:level>Info</ns1:level>
</ns:logWriteMessage>
```

The value for `<level>` matches one of the Server Logging Level properties on the Logging Properties page of Process Console. Your message is inserted in the log when an event occurs that is defined by that logging level. For example, storage exceptions are Critical level. Engine events, such as BPR deployments and server start up, are Info level.

The values for `<level>` include the following (case-sensitive):

- Info (default)
- Error
- Warning
- Critical
- Verbose (includes all levels)

In the Process Console, you can view the Server Log and filter Logging Service to show the Process filter. With the Process filter enabled, you can view all messages from all processes you create to add a server log comment.

Related Interfaces

System Services also includes two additional interfaces related to server logging.

- `ClearServerLog`
- `GetServerLogList`

Shell Command Invoke Service

1. In the Project Explorer, create a new orchestration project.

2. Select **File > New > BPEL Process**.

3. Name the process and click **Finish**.

4. In the Participants view, create a new Partner Service Provider.

5. Select System Services from the Interfaces tree.

6. Select *Shell Command*.

7. From the Partner Service Provider that you created, drag the **execCommand** operation to the canvas to create an invoke activity.

8. Fill in the input data, with the selections described above.

For more information, see *Connectors > Shell Command Invoke Service Connector*.

C H A P T E R   3 5

Human Tasks

This chapter includes the following topics:

Getting Started with Human Tasks

Human Tasks is an extension to WS-BPEL 2.0 that includes human workflow activities within a BPEL process. In addition to receiving, replying, and invoking Web services, Human Tasks "invokes" a person to handle and complete a task. Similar to the behavior of an invoke activity, the person returns output data to the BPEL process.

Human Tasks is also based on the WSDL 1.1, WS-HumanTask 1.0 and WS-BPEL Extension for People (BPEL4People), Version 1.0 specifications.

Note: Human Tasks is currently available only using Process Developer with the on-premises version of Informatica Business Process Manager (ActiveVOS).

For details about the people involved in a BPEL process, see *Introducing Human Workflow into a BPEL Process*.

About the BPEL4People Specification

The *WS-BPEL Extension for People (BPEL4People), Version 1.0* Specification provides extensions to the OASIS WS-BPEL 2.0 Standard to enable human interactions and a model of human interactions that are service-enabled. This specification is currently accepted by the OASIS WS-BPEL Extension for People (BPEL4People) Technical Committee for discussion as adoption as an OASIS Standard.

For details, visit `http://www.oasis-open.org/committees/bpel4people/charter.php`.

Introducing Human Workflow into a BPEL Process

Human Tasks contains extension elements and an extension activity, called the People activity, which offer the following:

- Creating a task definition.
- Selecting people for working on or managing a task.
- Creating workflow requirements, such as deadlines and escalations.

A People activity uses a task definition and contains several other properties. The People activity can be linked to any BPEL activity or contained within a scope or other container. You can add as many People activities to a BPEL process as needed.

See also *Routing Tasks to People at Run Time*.

Routing Tasks to People at Run Time

The following illustration shows what happens when a BPEL process executes a People activity.
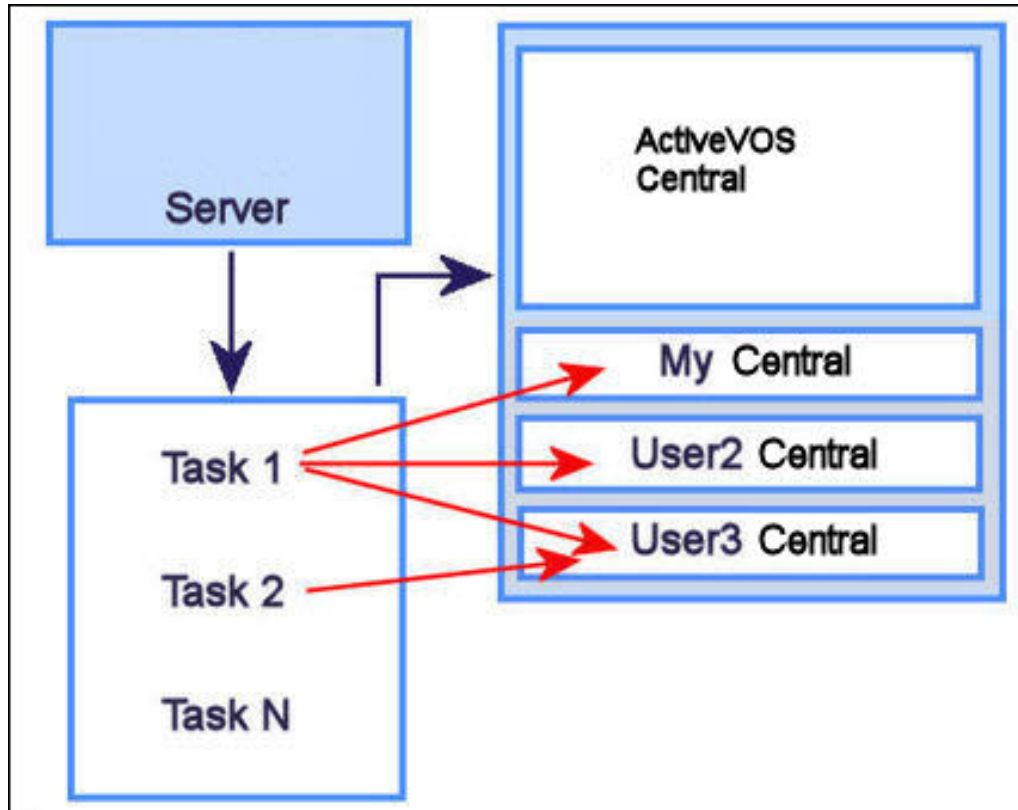
When a BPEL process executes a people activity, the task defined by the activity is routed to all potential owners and administrators defined by the task. Each user can claim a task to work on, making it unavailable for other users.

When the task owner completes the task by submitting the required information into a form, the output data is sent back to the process so that the next activity can execute.

Typically a BPEL process containing a People activity is an asynchronous, long-running process. Since the people activity effectively "invokes" a person, the expected response time is high. For this reason, when you

design a BPEL process containing a people activity, you should make the process asynchronous to avoid probable Web service communication timeouts.

About Task Life Cycle

The following are the basic states of a task after a BPEL process begins executing a People activity:

- **Unclaimed**. The task is available for anyone designated as a potential owner.
- **Claimed**. One person reserved and started the task. If only one user is assigned to the task, it is in the Claimed state automatically.
- **Completed**. The output data was submitted and the owner declared the task complete.
- **Failed**. The owner provided fault data and failed the task.
- **Exited**. An error condition occurred, such as expiration or People activity termination.

Creating the Artifacts Needed for the People Activity

The People activity must route a task to the appropriate group of people. Doing this requires the following artifacts:

- **WSDL** file that contains the port type and operation to "invoke" a person. The person receives the input data needed to make a decision in order to complete the task and reply with output data.
- An *identity service*, which includes the groups and users available for tasks. As an option for choosing a task owner, you can define an identity service on the Process Developer Preferences page.
- Informatica Business Process Manager includes an application to present task details to users. You can customize the presentation.

Participants and Tasks

To include human workflow activities in a BPEL process, you need a People activity that describes who works on a task. The Participants view is a key starting point for creating the people and tasks to be used in People activities. The human task participants listed in the Participants view correspond to BPEL4People's concept of a Logical People Group, as described in *About Logical People Groups*. The purpose of selecting a name is to give your Logical People Group a meaningful name. When you create a Process Deployment Descriptor for the process, you must select the actual names from the Identity Chooser.

If desired, you can select the actual group name from your company's LDAP or other identity service for a human task participant. Do this by selecting the button at the end of the Name field in a Logical People Group's Properties view.

Note that you must configure and enable an Identity Service before selecting from the Identity Chooser.

Tip: You can wait until deployment to map one or more groups from your Identity Service to a Logical People Group.

Creating a Human Task Participant

Creating a Human Task Participant

A human task participant represents the role of a person who performs tasks within the process. The participant is a Logical People Group that is used in building a task to be used in a People activity. It consists of a name and optional parameters.

You can add a task as a child of a participant.

To create a new human task participant:

1. In Participants view, right-mouse click on **Human Task Participants** and select "New Human Task Participant."

2. The Properties view opens for the new participant, and the default **Name** of the Logical People Group is "Role1".
   In a Logical People Group, people are assigned to a role such as Potential Owners or Business Administrators.

3. If desired, rename this group. Alternately, click the button at the end of the Name field to open the Identity Chooser.

Creating a Task for a Participant

After you create a human task participant, you can create a new process-level task for the participant.

Note: To create a new Notification, see *About Notifications*.

To Create a New Task:

1. In Participants view, right-mouse click on a Logical People Group name (such as "Role1"), and select **New Human Task**.

2. The default Task Name is "Task1". If desired, rename this to something more meaningful.

3. Do one of the following

   - Click to accept the name and to finish the details later in the Properties view.

   - Continue to the next step to select or create an interface.

4. To select an interface that you have already imported into the workspace, use the Interface tree. Select the WSDL operation for this task.
   The tree shows several categories: the current BPEL process and the WSDL interfaces currently in use for other participants (if any), Project and Project Reference Services, and Other workspace orchestration projects containing WSDLs (if any). For details on Using Project References, see the *Process Developer Online Help*.

5. To create a new interface, select **Generate Interface**. See "Creating a New Interface" in the *Process Developer Online Help* for details.

You will assign the Logical People Group is to the Potential Owners role in the task that is created.

Using or Mapping Users or Groups From the Identity Service

If desired, you can name actual users or groups from your Identity Service to the PDD Logical People Group panel. Note that the WS-HT specification does not allow a mixture of users and groups.

If you selected a user or group name from the Identity Chooser when you created the Logical People Group, you can use that name in the PDD.

If your Logical People Group has a generic name, such as "Role1", you can map this name to an actual user or group in your Identity Service.

Before you can select actual groups, you must ensure that your Identity Service is correctly enabled, as described in *Setting up an Identity Service*.

To add individual users, select the Users radio button, and then select **Add**. In the **Add User** dialog box, type in the name of one user, named exactly the same as in the Identity Service. For example, if you are testing, you may have defined a user in the `tomcat-users.xml` file as:

```
<user name="testUser1" password="b4p" roles="abTaskClient,
Finance,Marketing,Accounting,NERegionReps" />
```

In this example, you can type in `testUser1` as a User in the PDD Editor.

To add Groups, select the Groups radio button, and then select **Add** to add a new group name. You can also select from an Identity Chooser, as described in *Using the Identity Chooser During Deployment*.

Creating a Task for a Participant

Create the interface for a new BPEL for People task.

After you create a human task participant, you can create a new process-level task for the participant.

Note: To create a new Notification, see *About Notifications*.

To Create a New Task:

1. In Participants view, right-mouse click on a Logical People Group name (such as "Role1"), and select **New Human Task**.

2. Rename the default task name.

3. Do one of the following

    • Click to accept the name and to finish the details later in the Properties view.

    • Continue to the next step to select or create an interface.

4. To select an interface that you have already imported into the workspace, use the Interface tree. Select the WSDL operation for this task.
    The tree shows several categories: the current BPEL process and the WSDL interfaces currently in use for other participants (if any), Project and Project Reference Services, and Other workspace orchestration projects containing WSDLs (if any). For details on Using Project References, see the *Process Developer Online Help*.

5. To create a new interface, select **Generate Interface**. See "Creating a New Interface" in the *Process Developer Online Help* for details.

You will assign the Logical People Group is to the Potential Owners role in the task that is created.

Using BPEL4People Extension Elements and Activities

BPEL4People is an extension to the BPEL language and is incorporated into the Process Developer through extension elements and activities, namely:

• *Human Interactions Extension Element*
  This element is added to the Outline View and contains the child elements available for building a People activity, including Logical People Groups, Tasks, and Notifications.

• *People Activity*

Human Interactions Extension Element

Human Interactions is an element in the BPEL4People namespace that is an extension to BPEL. This element contains the child elements you can use to build a People activity, namely Logical People Groups, tasks and notifications.

You can add a Human Interactions element to the process or to a scope. As you define Logical People Groups, tasks and notifications at the process or scope level, they become available to the People activities at that level. Using a Human Interaction is a flexible and optional choice. The benefit is that the people assignments, tasks, and notifications within it are available to all People activities in a process.

The Human Interactions element is automatically added to the process when you use the *Participants* view to create people and tasks.

To manually add a human interactions element to the process:

1.   From the Outline View, right mouse click on the process name.

2.   Select **Add > Human Interactions**.

3.   The Outline View displays the BPEL4People elements you can use to build one or more People activities.

Tips:

-   Use the Logical People Groups, tasks, and notifications to define generic elements that you can use in one or more People activities. You can override some of the general settings in each People activity.

-   Add a Human Interactions element to a scope to create local definitions.

-   It is optional to define a human interactions element. You can define a task or notification inline within a People activity, for use only within that activity.

About Logical People Groups

Add Logical People Group parameters.

A *Logical People Group* is a named list that represents users or groups defined in an Identity Service, such as your company's LDAP directory. The Identity Service must be configured and enabled in the Process Console. Instead of naming the actual users or groups who can manage processes and tasks, you can refer indirectly to them from a Logical People Group.

The Logical People Group, such as *customer service representatives*, can have a parameter, such as *regions*, that can be associated with an expression, such as *region one*. Parameters can be used in an assign activity or with a people query at deployment time (for a JDBC or LDAP Identity service). At run-time, the parameters get resolved.

You can use Logical People Groups in defining task and notification assignments.

For example, you can define a Reviewers Logical People Group as a global construct and then use that Logical People Group within one or more tasks requiring reviewers. The users or groups are actually selected at deployment.

Logical People Groups are global elements enclosed in a Human Interactions extension element. The XML syntax for a Logical People Group element is:

```
<htd:logicalPeopleGroups>?
    <htd:logicalPeopleGroup name="NCName" reference="QName"?>+
        <htd:parameter name="NCName" type="QName" />*
    </htd:logicalPeopleGroup>
</htd:logicalPeopleGroups>
```

The `parameter` element is used to pass data needed for a dynamic assignment of a Logical People Group in an assign activity or for a role assignment within a task. A parameter can also be resolved with a people query evaluation at deployment for JDBC and LDAP Identity services, but not file-based services, such as

tomcat-users.xml. You can define several parameters of which some, none, or all of them can be evaluated and used in assignments or in the PDD. In addition, the parameters can be overridden in the Process Deployment Descriptor file.

Logical People Groups can be defined either at the process level or on enclosed scopes.

Creating a Logical People From the Outline View

Add Logical People Group parameters.

You can add a Logical People Group from the Outline View, and it will automatically be added as a Human Task Participant in the Participants View. For a key starting point to create Logical People Groups, see *Creating a Human Task Participant*.

Creating a Logical People Group From the Outline View:

1. From the Outline View, right mouse click on the process name.

2. Select **Add > Human Interactions**.

3. Expand the Human Interactions element to see Logical People Groups.

4. Right mouse click on Logical People Groups and select **Add > Logical People Group**.

5. In the Properties view, fill in the definition as follows:

    a. Type in a unique name, for example, "Regional Reps."

    b. Add parameters as desired. A parameter can be used in an assign activity or a people query for LDAP or JDBC Identity services in the Process Deployment Descriptor file. Its value can further be expressed by an argument within the role assignment in a task definition. For example, potential owners from the "Northeast Region."

Tip: You can create Logical People Groups at the scope level for use within the enclosing scope.

Logical People Group Parameters and Arguments

Add Logical People Group parameters.

A Logical People Group defined within the Human Interaction element can include parameters. They can be evaluated for specific people queries into an Identity service.

In addition, for each Logical People Group used in a task role assignment, you can add an argument expression for the parameter for that role assignment. When you associate the parameter with a search attribute in your LDAP or JDBC Identity service, the argument is resolved.

When you define a parameter, you can use it as follows:

- In an assign activity, create an expression for a parameter when copying one Logical People Group to another. For example, you can write an XPath or XQuery expression that reflects the following logic: "The Logical People Group for the next task is the previous task owner's manager."

- In a people query in the Process Deployment Descriptor, associate the parameter with a search attribute in your JDBC or LDAP Identity service. For example, associate the parameter `country_code` with an Identity search attribute called `country_code`.

- In a role assignment within a task, create an expression for the parameter for a more specific people query in a JDBC or LDAP Identity service (not a file-based Identity service). As the parameter is resolved at runtime, the role assignment expression is also evaluated.

Example: Parameters in a Logical People Group Definition

```
<htd:logicalPeopleGroup name="regionalClerks">
   <htd:documentation xml:lang="en-US">
     The group of clerks responsible for the region specified.
   </htd:documentation>
```

```
            <htd:parameter name="region" type="xsd:string" />
        </htd:logicalPeopleGroup>
```

Example: Arguments in Logical People Group Definition for a Potential Owner of a Task

```
<htd:potentialOwners>
    <htd:from logicalPeopleGroup="regionalClerks">
      <htd:argument name="region">
        htd:getInput("part1")/region
      </htd:argument>
    </htd:from>
</htd:potentialOwners>
```

Using a Logical People Group in an Assign Activity

For dynamic assignment of a Logical People Group, you can create Copy Operations in an Assign activity. A Logical People Group can be copied as follows:

Copy from a Logical People Group to:

- Another Logical People Group

  - Variable or variable property

  - Copy to a Logical People Group from:

- Literal

  - Expression

  - Variable or variable property

XML Syntax

```
<wsht:from logicalPeopleGroup="NCName">
  <wsht:argument name="NCName" expressionLanguage="anyURI"?>*
    value
  </wsht:argument>
</wsht:from>
<wsht: to logicalPeopleGroup="NCName"/>
```

Example: From a Logical People Group Including a Parameter to a Process Variable

```
<bpel:assign name="getVoters">
  <bpel:copy>
    <wsht:from logicalPeopleGroup="voters">
      <wsht:argument name="region">
        $electionRequest/region
      </wsht: argument>
    </bpel:from>
    <bpel:to variable="voters" />
  </bpel:copy>
</bpel:assign>
```

Example: From a Literal to a Logical People Group

```
<bpel:assign>
   <bpel:copy>
    <bpel:from>
      <bpel:literal>
       <myns:entity xsi:type="htd:tOrganizationalEntity">
        <htd:users>
         <htd:user>Elaine Smith</htd:user>
         <htd:user>Julie Delgado</htd:user>
        </htd:users>
       </myns:entity>
      </bpel:literal>
     </bpel:from>
    <wsht:to logicalPeopleGroup="Directors" />
   </bpel:copy>
</bpel:assign>
```

Example: From an XQuery Expression to a Logical People Group

```
<ns3:copy>
   <ns3:from expressionLanguage="urn:active-endpoints:expression-language:xquery1.0">
    <ht:organizationalEntity xmlns:ht='http://www.example.org/WS-HT'>
      <ht:users>
        <ht:user>{$loanOfficers/loanManager}</ht:user>
      </ht:users>
    </ht:organizationalEntity></ns3:from>
   <ns3:to b4p:logicalPeopleGroup="loanreps"/>
</ns3:copy>
```

Selecting Users or Groups for Logical People Groups During Deployment

Selecting Users or Groups for Logical People Groups During Deployment

During deployment, you can select list of users or groups or create a query to select users or groups for each Logical People Group defined in the process. The users or groups are those defined in your organization's Identity Service, such as a Tomcat Users file (for testing) or an LDAP or JDBC directory.

The user or group assignments you make are based on the `hd:organizationalEntity` data type, defined in the WS-HT specification.

Actual assignment of people to roles occurs when a process is instantiated. This runtime behavior allows members of a role to be determined on a just-in-time basis and is beneficial when group membership changes frequently.

Using Logical People Groups for Role Assignments

Select the value of the Role assignment: literal, expression, or Logical People Group.

A Logical People Group is a construct that represents the users or groups who perform a role related to a task or notification. By defining one or more Logical People Groups, instead of naming actual users or groups, you can re-use a Logical People Group in a variety of ways.

For details on creating a Logical People Group, see *About Logical People Groups*.

To assign a Logical People Group to a task or notification-related role:

1. From the Outline view, select a Human Interactions element or People activity containing a task or notification.

2. Select a task or notification to put the Properties View in focus.

3. On the Assignment tab, select a role. For example, select Potential Owners. For definition of roles, see *Role Assignments for a Task or Notification*.

4. In the Value column, click (...).

5. In the **Role** dialog, select Logical People Group as the Type.

6. Select a name from the Group list.

7. If the Logical People Group is defined with parameters, you can do the following:

    a. In the Arguments panel, select a parameter.

    b. Click (...) in the Expression column.

    c. In the Argument Expression Builder, create an expression for the parameter. For example, if the parameter is *regions*, create an expression that will resolve to *region one*. For related examples, see *Using Expressions for Role Assignments*.

Note: You can only use parameters and arguments in an assign activity or in a people query in the Process Deployment Descriptor file. The people query is restricted to JDBC or LDAP Identity services.

For details and examples, see *Logical People Group Parameters and Arguments*.

Using Literal Values for Role Assignments

Select the value of the Role assignment: literal, expression, or Logical People Group.

Use the Literal option to directly specify users or groups for task and notification-related people assignments. Using Literal, you generate an XML document with placeholder values that you can fill in for individual users or groups.

You cannot have a mixed list of users and groups, according to the WS-HT Human Task specification.

To Add a Literal Value for a Task or Notification Role:

1. From the Outline View, select a task or notification.
2. In the Assignment tab of the Properties view, select a role, such as a task role of *potential owners*.
3. In the Value column, click (...).
4. In the **Role** dialog, select Literal from the Source list.
5. Select **Generate**.
6. Notice that Process Developer generates the `ht:organizationalEntity` element. This element is required when defining people assignments.
7. In the text box, modify the values to add users or groups.

The following literal contents are generated:

```
<ht:organizationalEntity xmlns:ht='http://www.example.org/WS-HT'>
  <ht:users>
      <ht:user>Some User</ht:user>
  </ht:users>
</ht:organizationalEntity>
```

Modify this XML sample as follows:

*   Delete or fill in valid values for each `Some User` in `<ht:users>`. The user name must match a User Id in your organization's directory, such as `MyFirstName.MyName`. Add additional `<user>` elements as desired.
*   Replace `<users>` with `<groups>`. Add elements and valid values for `<ht:group>`.
*   According to the WS-HT specification, you cannot mix users and groups

The following is an example of a valid XML data sample:

```
<htd:potentialOwners>
    <htd:from>
      <htd:literal>
        <htd:organizationalEntity>
          <ht:users>
            <ht:user>Ann.Weston</ht:user>
            <ht:user>Anu.Martin</ht:user>
            <htd:user>Deborah.Smile</ht:user>
          </ht:users>
        </ht:organizationalEntity>
      </ht:literal>
    </ht:from>
</ht:potentialOwners>
```

Using Expressions for Role Assignments

Select the value of the Role assignment: literal, expression, or Logical People Group.

Use the Expression option to write a conditional expression for selecting task or notification-related roles. The expression can be based on the input data of a People activity. For example, if you are invoking a service to retrieve identities, you can pass them into the potential owners or administrators identities.

The expression can include a process variable or it can be composed using WS-HT custom XPath functions, described in *Custom Functions*.

To Add an Expression for a Task or Notification Role:

1. From the Outline View, select task or notification.

2. In the Assignment tab of the Properties view, select a role, such as *potential owners*.

3. In the Value column, click (**...**).

4. Select Expression as the Source, and then select **Expression Builder**.

5. In the Expression Builder, select a Human Task (WS-HT) custom function or other variable in creating an expression.

6. Select the users or groups for the role.

XML Syntax

```
<htd:from expressionLanguage="anyURI"?>
    expression
</htd:from>
```

Example: Expression for a Task's Role Assignment

```
<htd:potentialOwners>
    <htd:from>$voters/users/user[i]</htd:from>
</htd:potentialOwners>
```

The potential owners definition is contained within a For Each activity.

Example: Using a WS-HT Custom Function:

```
<htd:potentialOwners>
    <htd:from>
        htd:getInput("part1")/approvers
    </htd:from>
</htd:potentialOwners>
<htd:businessAdministrators>
    <htd:from>
        htd:except(htd:getInput("part1")/admins,
        htd:getInput("part1")/globaladmins[0])
    </htd:from>
</htd:businessAdministrators>
```

The WS-HT custom function `getInput` returns a list of potential owners defined as the child element `approvers` in `part1` of the task's input message. The business administrators group includes `globaladmins` and excludes `admins`.

Example: A Notification's Role Assignment

```
<htd:peopleAssignments>
  <htd:recipients>
   <htd:from>
     htd:getPotentialOwners("ApproveClaim")
   </htd:from>
  </htd:recipients>
</htd:peopleAssignments>
```

Example: XQuery

```
<htd:potentialOwners>
 <htd:from>
  <htd:literal>
   <ht:organizationalEntity
     xmlns:ht='http://www.example.org/WS-HT'>
    <ht:groups>
     <ht:group>{data($creditInformation/loan:loanTypeGroup)}
     </ht:group>
    </ht:groups>
   </ht:organizationalEntity>
     </ht:literal>
```

```
        </ht:from>
    </ht:potentialOwners>
```

About Tasks

To include human participation as part of process execution, you can add a People activity and define a task for it.

A task is a Human Task (WS-HT) construct that relies on a WSDL interface, people assignments, and other details. By itself a task is not executable. It is designed to be used by a People activity. A special type of task is a notification. For details, see *About Notifications*.

The BPEL4People specification defines several ways that People activities can incorporate tasks, as shown in the following illustration.
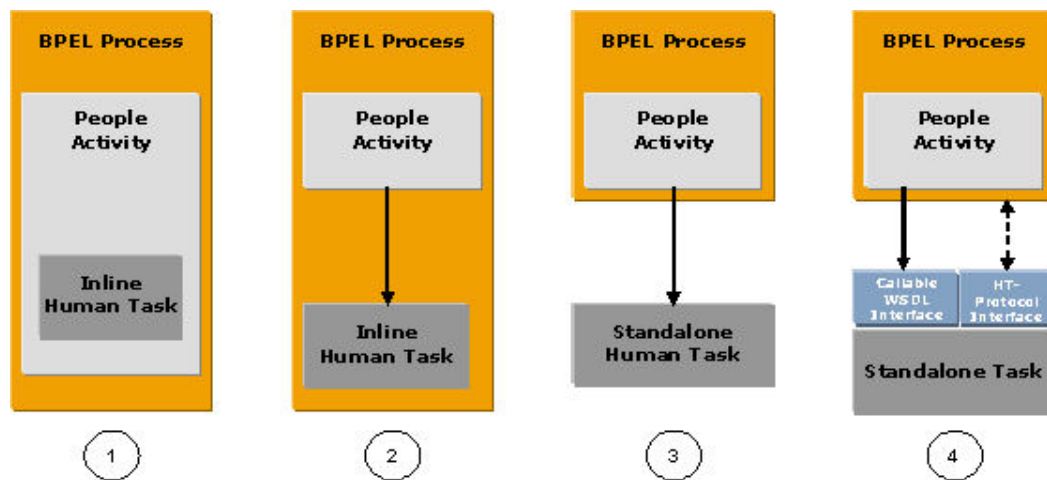
Process Developer handles each of the task implementations as follows:

- **Inline Task #1.** Define a task within a People activity for use only within that activity
  For implementation details, see *Creating an Inline or Local Notification Action for a Task Deadline*.

- **Inline Task #2 (Local Task).** Define a task within the Human Interactions element on the Outline view to make it available to all People activities in a process. The People activity can use the task as is or create overrides for some properties.
  For implementation details, see *Adding a Task or Notification to the Outline View for Process or Scope Use*.

- **Stand Alone Task #3**. (This is a future implementation.) Import into the process a human interaction XML document that contains a task definition and refer to the document within a People activity.

- **Stand Alone Task #4**. (This is a future implementation.) Refer to a remote task.

A task definition includes the following parts:

- Task Name, Interface, and Priority

- Role Assignments for a Task or Notification

- Adding Task or Notification Presentation Properties

- About Task Deadlines and Escalations

- Using Expressions for Outcome and Search By

- Adding Rendering Details for a Task or Notification

See also *Required and Optional Properties for a Task* and *Using the All Tab of a Task*.

Adding a Task or Notification to the Outline View for Process or Scope Use

To make a task or notification available for all People activities within a process or scope, you can create definitions within the Human Interactions element on the Outline view.

1. In the Outline view, right-mouse click on the process name or a scope activity.
2. Select **Add Human Interaction**.
3. Expand "Human Interactions" and right-mouse click on **Tasks** (or Notifications).
4. Select **Add > Human Task**.The Human Task wizard opens, where you can select a WSDL interface or select **OK** to complete the WSDL details later.
5. In the Properties view for the task, fill in the details.

A task definition includes the following parts:

- Task Name, Interface, and Priority
- Role Assignments for a Task or Notification
- Adding Task or Notification Presentation Properties
- About Task Deadlines and Escalations
- Using Expressions for Outcome and Search By
- Adding Rendering Details for a Task or Notification

For required properties, see Required and Optional Properties for a Task.

A notification definition includes parts described in About Notifications.

Required and Optional Properties for a Task

The following table describes required and optional properties required to make a task valid as well as properties you can optionally define:

Required Properties	Optional Properties
Interface	Outcome and Search By Expressions
Priority	Task Deadlines - All
People assignments	Renderings
Subject	Description

Task Name Interface and Priority

In the Properties view of a task, the **Task** tab displays the following properties:

- **Name**. (Required). The name combined with the target namespace of a task element uniquely identifies the task definition. This is not the name displayed to task users; that name is Display Name, as described in *Adding Task or Notification Presentation Properties*.
- **Priority** (Required). You can set a default priority level as a non-negative integer, starting at zero as the highest priority. You can create the priority as an expression. The priority is displayed in to users, and task owners or administrators can change the level. A way to raise the priority when the task is in progress is to set an escalation action, described in *About Task Deadlines and Escalations*. Note that priorities 0 and 1 are displayed with red and yellow flags, respectively. For an illustration of how a priority appears to users, see *Adding Task or Notification Presentation Properties*.

- **Interface**. (Required). Specifies the WSDL port type and operation used to invoke the task. The WSDL must be available in the Project Explorer. See *Adding a Task Interface*.

See also *About Tasks*.

Adding a Task Interface

Enter an interface for a callback when a task is described with a one-way operation.

A task is based on a WSDL port type definition. Essentially, a task represents a "service" that is 'invoked' by an operation that contains input and optionally, an output and fault message. You must import a WSDL into your process and then select the port type and operation as an interface to the task. The operation must be a request-response type.

If your process already contains an import for the WSDL that defines the Port Type and Operation, then select the appropriate values.

Role Assignments for a Task or Notification

Specify the constraints for task delegation.

In the Properties view of a task, select the Assignment tab to define participants for the generic roles associated with a task or recipients for a notification.

For a task, you must select at least one user as a potential owner, business administrator, or task stakeholder. For a notification, at least one recipient is required.

If you assign a single user to a task, the task is in auto-claim mode. When the assigned user logs into Process Central, the task is already in the Claimed state. Normally the task is in the Ready state for a potential owner to claim it.

The generic roles are as follows:

Role Assignment	Description
Potential Owners	Individuals or groups of people who receive a task. Each person receives the same task. Any one of them can claim ownership of a task, making it unavailable for all others.
Excluded Owners	Individuals or groups of people who may not become an actual or potential owner and thus they may not reserve or start the task.
Task Stakeholders	Individuals or groups of people who can influence the progress of a task, for example, by adding attachments or simply observing the state changes of the task. They are also allowed to perform administrative actions on the task instance and associated notifications, such as resolving missed deadlines.
Task Initiator	The person who creates the task instance. Depending on how a task is instantiated, a task initiator may or may not be defined. A case where the initiator could be known is when the actual owner or business administrator of Task A initiates Task B. At runtime, a task initiator may skip the task, if it is enabled as "skippable." The task appears in the initiator's task view with a Skip button. For details, see *Required and Optional Properties for a Task*.

Role Assignment	Description
Business Administrators (Task and notification)	Individuals or groups of people who are responsible for ensuring that the task or notification is completed. As such, Administrators can assign a task to any user or claim a task for themselves in order to work on it. An Administrator can also see a list of all task instances for all potential and actual owners of a task. When you set up task deadlines, you can trigger events, such as notifications, that can be targeted for Administrators. Administrators have the same permissions as task stakeholders.
Delegation (Task only)	In addition to selecting individuals or groups to receive the task, you can specify constraints concerning delegation of the task to others. By default, there is no constraint. Potential owners, the actual owner or a business administrator can delegate a task to any other user. On the All tab of a task, you can select anybody, **nobody** or potential owners. You can also specify individual or groups of delegatees by creating a literal, expression, or logical people group.
Recipients (Notification only)	Individuals or groups of people who receive notifications.

To define one of the generic roles, see:

- Using Literal Values for Role Assignments
- Using Expressions for Role Assignments
- Using Logical People Groups for Role Assignments

Adding Task or Notification Presentation Properties

Adding Task or Notification Presentation Properties

You can add information and instructions for task owners and administrators to help them work on the task. For a notification, there is no work to do, so typically detailed information is not required.

As you are creating the presentation properties, keep in mind how they will appear to users. The following illustration shows an example task for a user, and the list below describes the user interface elements you can create within the Presentation tab of a task or notification.
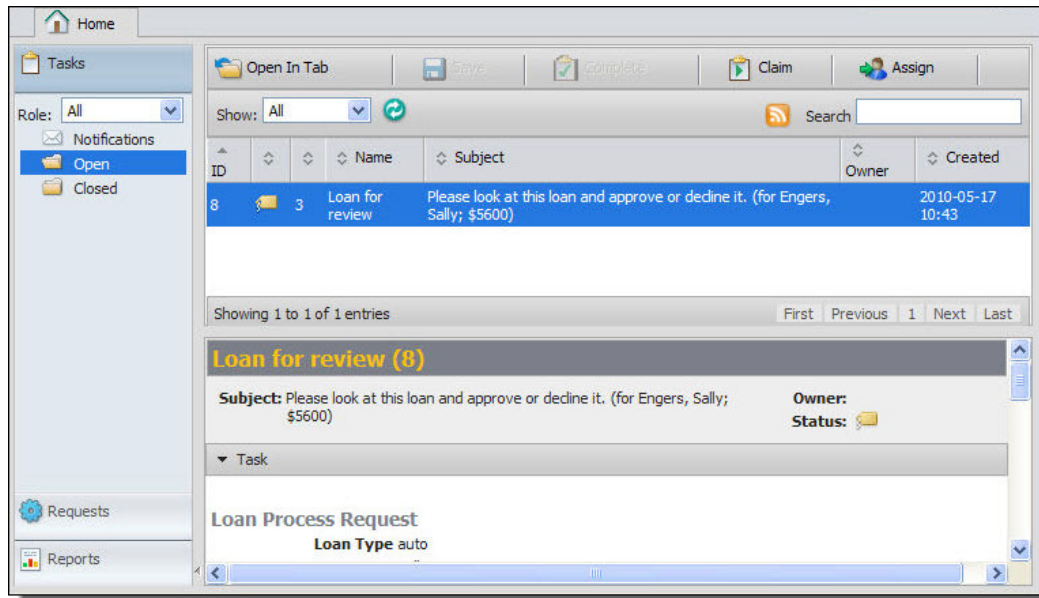
- **Priority** (Default). A task or notification has a priority level, displayed as an integer, starting at zero as the highest priority. (For details on creating a priority, see *Task Name, Interface, and Priority*.) In the illustration above, the task priority of 3 is shown to the left of the task Name.

- **Display Name** (Optional). The property appears as the Name to users. The maximum length is 64 characters.

- **Subject** (Required). Short description of the task or notification. You can write this as described in *Configuring Presentation Parameters for a Task or Notification Subject or Description*. The maximum length of the resulting string is 254 characters.

- **Detailed Description** (Optional). Long description of the task or notification as seen in the Additional Information section. You can write this as described in *Configuring Presentation Parameters for a Task or Notification Subject or Description* and *Adding a Content Type for a Task or Notification Description*.

- **Renderings** (Optional). For details, see *About Task Presentation in Process Central*.

Configuring Presentation Parameters for a Task or Notification Subject or Description

Provide presentation details for the task or notification.

The task Subject property defines the basic information for users to know what the task is all about. The subject line is similar to an email subject line, and is displayed similarly.

The task Description property can be lengthy and is displayed in the Additional Information section of a task.

You can specify the subject and description by using presentation parameters, which are expression-based variables that can be included in text. For example, a subject might be composed as:

```
Approve the insurance claim for {$lastname}, {$firstname}.
```

In this example the parameters, enclosed by curly braces, are configured by referring to child elements in an input message.

Presentation parameters can be based on input data unique to a task instance, making each subject line or description unique.

See the example in topics that follow.

In addition, you can add multi-lingual (localization) support for a Subject. For details, see *Adding MultiLingual Support for Task Subject, Description, and Display Name.*

You can also use Presentation Parameters to create custom task properties. For details, see *Creating Custom Task Properties*.

To Configure Presentation Parameters:

1. Put the Presentation tab of a task in focus in the Properties view.

2. Select the Dialog button at the end of either the Subject or Description field.

3. In the **Presentation Parameters** dialog, select **Add**. The placeholders `param` and `string` are added to a row.

4. Type over `param` with a meaningful **Parameter** name, such as the name of an input message part you will use in the parameter expression. For example, type in `lastname`.

5. Select the parameter Type from a drop-down list. The type is a simple XSD type. The type `string` is selected by default.

6. Click (...) in the *Expression* column.

7. Create an expression in the Expression Builder.
   Tip: Use one of the BPEL4People custom functions to return data from a task input message. See the example below.

The following illustration shows an example of presentation parameters within the Subject.
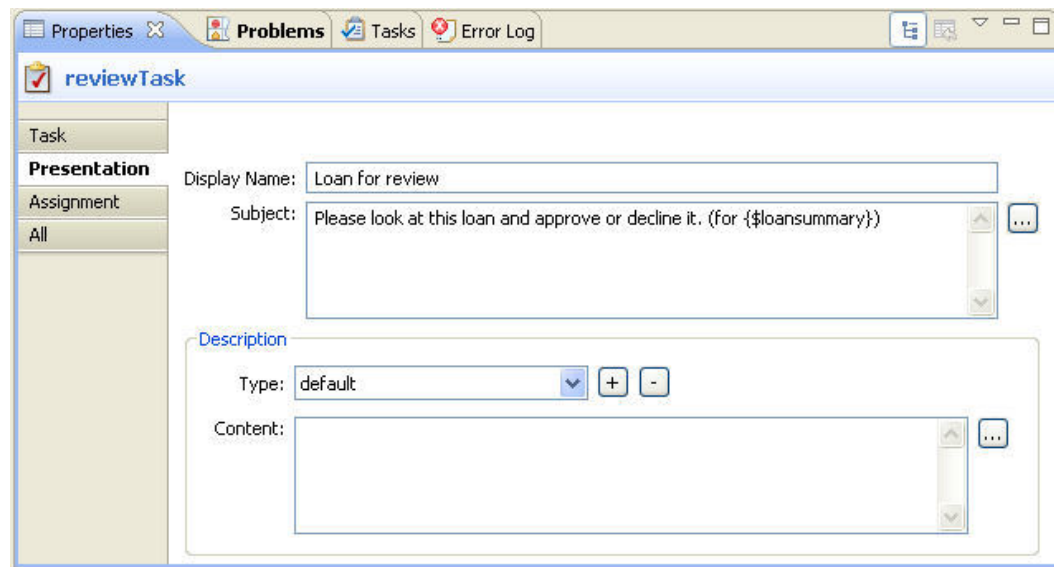
Adding MultiLingual Support for Task Subject Description and Display Name

On a task's Presentation tab, you can externalize the text in a Subject's expression, a Description's expression, and the Display Name. For an overview of how to create an expression, see *Configuring Presentation Parameters for a Task or Notification Subject or Description*.

Externalizing the Text Strings for a Subject, Description, and Display Name:

1. Open the Properties view of a task and select the Presentation tab.

2. At the end of the **Subject** expression that you have added, append the required syntax for multilingual support, as follows:
   `| [keyname, {$var}]`

where:

- "|" (vertical bar) is the separator
- "[ ]" (brackets) contain all externalization values
- `keyname` is an identifier for the Subject expression
- `{$var}` represents a variable used in the Subject expression

These are shown in the following figure:
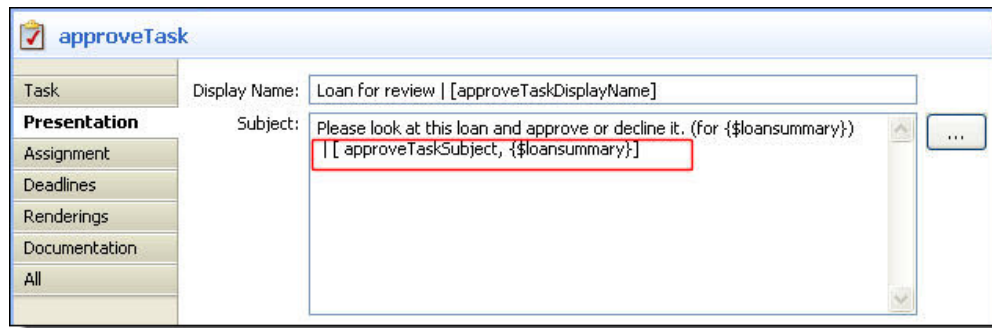
Here are some examples. Note that the default expression is to the left of the vertical bar and the externalization values are to the right:

- `This is my subject | [taskSubject]`
- `My subject is {$parameter} | [taskSubject, {$parameter}`]
- `My subject is {$param1} and {$param2} | [taskSubject , {$param1}, {$param2} ]`

3. Add the bundle key name and value to each message properties file that you create for multilingual support. For example, in the English file:

- `taskSubject=This is my subject`
- `taskSubject=My subject is {0}`
- `taskSubject=My subject is {0} and {1}`

For details on creating message property files, see "Adding MultiLingual Support to Process Central" in the *Process Developer Online Help*.

4. Repeat steps 2 and 3 for the task's Description. The Description appears in the Task Details section in Process Central.

5. Repeat steps 2 and 3 for a task Display Name. Note that only a key name is allowed.

6. Be sure that your Process Central configuration file (`.avcconfig` file) contains an i18n location reference for your default properties file. For details, see "Adding MultiLingual Support for an .avcconfig File" in the *Process Developer Online Help*.

Adding a Content Type for a Task or Notification Description

Add a content type for a description.

You can specify one or more content types, such as plain text, HTML, or XHTML, for the text in a task description. You can then compose the description differently, as desired, for each content type. The default content type is plain text.

Adding a Content Type:

1. Put the Presentation tab of a task in focus in the Properties view.

2. In the Description panel, select the Add (**+**) button next to Type.

3. In the **Content Type** dialog, type in a new content-type value. For example, add "text/html".

Note: Process Developer supports plain text, HTML, and XHTML. However, you can add any content type you wish that you may be supporting in your own client task application.

Using the All Tab of a Task

In a task's Properties view, the All tab displays the complete list of properties, including some that do not appear on the Task, Presentation, and Assignment tabs.

You can define or modify properties on the All tab or on the other tabs as desired.

For details on properties that appear only on the All tab, see the following topics:

*   Using Expressions for Outcome and Search By

*   Adding Rendering Details for a Task or Notification

*   Role Assignments for a Task or Notification

Using Expressions for Outcome and Search By

Create an expression for task outcome.

You can use the following two expressions:

*   **Outcome**
    This optional element identifies the field of an XSD simple type variable in the output message that reflects the business result of a task. Using Outcome enables you to create a query to produce an `xs:string` which represents the outcome of the task. This query is run against the specified part of the output data.

    This element is not displayed in an Process Developer task client, but can be useful as part of the HT API.

*   **Search By**
    This optional element lets you create an expression which produces an `xs:string` that can be used to quickly locate a task within the a task client application. The expression evaluates using the input data for the task. For example, a loan application request contains several data parts, including an application ID customer name, and other loan details. The Search By expression may extract the application ID from the input data. In a task client, a user can type in an application ID in the Search By field to quickly retrieve a particular task.

    The Search By expression created in Designer might be:

    ```
    htd:getInput('loanPart')/@applicationId
    ```

    where `application Id` evaluates to a five-digit number associated with the loan application request. In the Process Developer task client, a user would enter a five-digit number in the Search By field, such as "12345", to locate a particular task.

About Task Deadlines and Escalations

Select a deadline event (escalation) to view its definition.

You can specify one or more deadlines by which a task must be started and also one or more deadlines by which a task must be completed. If a start date or completion date is missed, you can trigger one or more escalation actions. For example, set a "start the task" deadline for three hours after a task is created, and if the deadline is missed, send a notification to the task's owner. If the second deadline is missed, alert the business administrator.

In addition to the simple example of notifying a task owner when a task is late, there are many expressions and conditions you can apply to a task deadline.

Note the following about setting deadlines:

- Start deadlines describe when a potential owner must start working on a task. The deadlines are enabled as soon as the task is created and are disabled after a potential owner starts the task.

- Completion deadlines describe when an owner must complete a task. The deadlines are enabled as soon as the task is created and are disabled after an owner completes the task.

If a task is reassigned, which changes the task state to unclaimed, the original deadlines are still in effect. For details, *see About Task LifeCycle*.

See the following topics for details on setting deadline alarms and escalation actions.

- Adding Start and Completion Deadlines and Alarms

- Selecting an Escalation Action to Occur When a Task Deadline is Triggered

- Defining a Condition for an Escalation Action

- Using Task Data or Mapped Data for a Notification

- Creating an Inline or Local Notification Action for a Task Deadline

- Using a Reassignment Action for a Task Deadline

- Invoking a Process When a Task Deadline Occurs

Adding Start and Completion Deadlines and Alarms

You can set deadlines for when a task must start and when a task must complete.

If desired, you can set deadlines for when a task must start and when a task must complete. For example, if a task must start within three hours of being created, you can set a start deadline and add an escalation action to notify the potential owners that no one has started the task. The notification is sent to all potential owners.

You can add multiple start and completion deadlines for a task. For details on the definition of start and completion, see About Task Deadlines and Escalations.

Creating a Start (or Completion) Deadline for a Task:

1. In the Outline view, select a people activity that has a task (not a notification).

2. Select the people activity's task.

3. In the task Properties view, select the Deadlines tab, and select **Add**.

4. In the **Deadline Specification** dialog, select the deadline type: Start or **Completion**. A start deadline indicates when a task must be started by. A completion deadline indicates when a task must be completed by.

5. Select an Alarm Type: Deadline or Duration. You can fill in the expression later.

6. Select one of the escalation types to occur when the deadline occurs:

    - inline notification

    - local notification

    - reassignment

    - process invoke

    - send email

7. In the table, click (**...**) in the *Expression* column to create a deadline or duration value that conforms to the type selected.

8. To complete the details for the escalation, see Selecting an Escalation Action to Occur When a Task Deadline is Triggered.

9. Add more start and completion deadlines as desired.

Tips:

- The start time begins when the task is created; that is, when the BPEL process executes the People activity.

- An example of a duration of six hours is `PT6H`. For more examples on how to express deadlines or durations, refer to the *Process Developer Online Help*.

- If desired, add a condition to the expression.

- On the **Deadlines** tab of a task, select a deadline and then select **Go To** to view the properties of the deadline.

Selecting an Escalation Action to Occur When a Task Deadline is Triggered

You can set deadlines for when a task must start and when a task must complete.

If desired, you can define start and completion deadlines for a task, and when a deadline is reached, trigger an escalation action. For details on how to set a task deadline, see Adding Start and Completion Deadlines and Alarms.

An *escalation action* is an event that ranges from a simple notification to a process invocation.

The following table describes the escalation actions available to occur when you set a task deadline.

Escalation Action	Description
Inline Notification	Adds a new notification available only to the current task
Local Notification	Adds a notification that was defined in the process or enclosing scope
Reassign Task	Removes the task from the current users and groups' task view and reassigns it to specified users or groups
Invoke a process	Executes a BPEL process that may, for example, send an email instead of a notification
Send email	Sends an email to a list of recipients; this requires that the Email and Identity services are enabled and configured in Process Server

In addition, you can trigger an escalation action under certain conditions and with certain data. For details, see:

- Defining a Condition for an Escalation Action
- Using Task Data or Mapped Data for a Notification

Defining a Condition for an Escalation Action

As you define deadlines for the start and completion of a task, you can add a condition for triggering an escalation action. For example, you can check to see what the current priority of the task is, and if it is a high priority, trigger the escalation action. Escalation actions include notifications, reassignments, and process invocations, as described in Selecting an Escalation Action to Occur When a Task Deadline is Triggered.

If you add a condition to an escalation action, the escalation does not occur unless the condition evaluates to true.

Adding a Condition for an Escalation:

1.   Open the Properties view of an escalation, such as an inline notification under a Start Deadline of a task.

2.   Select the Dialog (**...**) Button at the end of the Condition text box.

3.   In the **Condition Builder**, select the functions and variables for your condition. For example:
     `htd:getInput("ClaimApprovalRequest")/amount < 10000`

In this example, the `amount` part of the `ClaimApprovalRequest` variable must be less than $10,000 in order for the condition to evaluate to true.

As you write conditions, you can take advantage of the Human Task functions, in particular the `getInput` function shown in the example. This function returns data from a running task instance.

Using Task Data or Mapped Data for a Notification

Notification data. Use task data or specify how to map data to another variable part.

Like a task, a notification has input data based on a WSDL interface. The interface for a notification serving as a task deadline's escalation action may be the same or different from the interface for the task.

The Properties view of an inline or local notification has a Data section:

*   **Use Task Data**. If the notification's input data is the same as the enclosing task's, then this option is available for selection. The input message of the task is passed to the notification.

*   **Map Data**. If the notification's data is a different type, then you can assign appropriate data to the notification to create a multi-part WSDL message from the data. The part attribute refers to a part of the WSDL message. You must map each part for every part in the WSDL message definition to avoid uninitialized parts in the target WSDL message.

To Map Data:

1.   In the Properties view of a notification used as an escalation, select the *Map Data* radio button.

2.   Select the **Dialog** button in the Expression row for the first message part.

3.   Create an expression for the data mapping, as shown in the following illustration:

Example

This example comes from the WS-HT specification:

```
<htd:toParts>
  <htd:toPart name="firstname">
    htd:getInput("ClaimApprovalRequest","ApproveClaim")
           /firstname
  </htd:toPart>
  <htd:toPart name="lastname">
    htd:getInput("ClaimApprovalRequest","ApproveClaim")
           /lastname
  </htd:toPart>
  <htd:toPart name="taskId">
         htd:getTaskID("ApproveClaim")
  </htd:toPart>
</htd:toParts>
```

Creating an Inline or Local Notification Action for a Task Deadline

A common escalation action associated with a task deadline is a notification. When a deadline is reached, a notification can alert a task owner about a late task. For a discussion of escalations, see About Task Deadlines and Escalations.

Two of the escalation actions you can select are inline notification and local notification.

- An *inline* notification is a new notification, defined exclusively, in this case, as an escalation action. For details on defining a notification, see About Notifications.

- A *local* notification is defined at the process or scope level, within a Human Interaction element. You can customize the notification for a particular escalation by providing overrides for the existing people assignments and the existing priority.

Creating an Inline Notification:

1. In the Outline view, right mouse click on **Task > Start (or Completion) Deadline > Inline Notification.**

2. In the Properties view, fill in all the required details for the notification, including interface, assignments, priority, and presentation Subject.

Creating a Local Notification:

1. In the Outline view, right mouse click on **Task > Start (or Completion) Deadline > Local Notification**.

2. In the Local Notification Properties view, select a Notification from the picklist.

3. If desired, add a new priority.

4. If desired, add new assignments on the Assignments tab.

For details on setting a condition and using task data, see Defining a Condition for an Escalation Action and Using Task Data or Mapped Data for a Notification.

Using a Reassignment Action for a Task Deadline

Reassignments are used to replace the potential owners of a task when an escalation is triggered. For a discussion of escalations, see About Task Deadlines and Escalations.

To Create a Reassignment:

1. In the Outline view, right mouse click on **Task > Start (or Completion) Deadline > Reassignment**.

2. In the Properties view, select potential owners for the task.

For details on adding a condition, see Defining a Condition for an Escalation Action.

Invoking a Process When a Task Deadline Occurs

The Invoke Process action is an extension escalation provided by Informatica. This action is based on Informatica-provided WSDL, "Custom B4P Notification", as described in Creating Custom Escalation Actions.

You can deploy a BPEL process to your server and then specify the service name as an escalation action. You might want to invoke a process to send an email to a user, rather than send a notification to a task client application.

To Create an Invoke Process Action:

1. Ensure that you have deployed your BPEL process that serves as a custom escalation action. This process must be based on the special template, Custom B4P Notification.

2. In the Outline view, right mouse click on **Task > Start (or Completion) Deadline>Process Invoke**.

3. In the Properties view, add the Service name of a BPEL file deployed to the same server where your current process will run. You can copy the service name from the Process Console's Service Definitions page.
   Note: You must use the special BPEL template, Custom B4P Notification, designed for a custom notification, as described in Creating Custom Escalation Actions.

4. If desired, create a Data expression for task or other data you wish to pass to the called service. For example, create an expression to determine the task priority or list of potential owners.

For details on adding a condition, see Defining a Condition for an Escalation Action.

Sending Email When a Task Deadline Occurs

The Send Email action is an extension escalation provided by Informatica. When a deadline is reached, an email can alert a task owner or administrator about a late task. For a discussion of escalations, see About Task Deadlines and Escalations.

Sending email requires that you set up the Email Service in Process Server. The Email service provides a default sender's email address, which you can configure. For details, see the *Process Server Online Help*.

To Create a Send Email Action:s

1. In the Outline view, right mouse click on **Task > Start (or Completion) Deadline > Send Email**.

2. In the Email tab, add an expression for one or more recipients. The assignment you make to recipients must evaluate to the `hd:organizationalEntity` data type, defined in the WS-HT specification. Typing the name of a Logical People Group or a user name is not a valid value. To create a valid expression, use a Human Task custom function from the Expression Builder, such as:
   `htd:getPotentialOwners("reviewTask")`

3. Add an expression for the Subject and for the Body of the email. To write these expressions, see the example below. Also, see Email Service in the *Process Developer Online Help* for some tips on creating the Subject and Body using expressions.

4. Optionally, add a condition, as described in Defining a Condition for an Escalation Action.

Here are examples of valid email values:

- **Recipients:** `htd:getLogicalPeopleGroup('loanreps')`

- **Subject:** `'New task in Central'`

- **Body:** `concat('Please review new loan for ', htd:getInput('Document')/loan:lastName, 'which has just arrived in Process Central.')`

About Notifications

A notification is a WS-HT construct that relies on a WSDL interface, people assignments, and other details that are incorporated into a People activity.

In BPMN, a notification is implemented within a Send Task or Message Send Event.

By itself a notification is not executable. It is designed to be used as an escalation action within a task or as the main construct of a People activity.

Unlike a task, a notification does not have an outcome. It relies on a one-way operation in a WSDL, serving to inform users of an event such as missed deadline or product ship date. Notifications usually are displayed in users' task client application along with tasks. They could also be used in other forms, like email or text message, if you choose to create your own client applications.

The BPEL4People specification defines several ways that a People activity can incorporate a notification, as follows:

- An inline notification declared within the People activity. The notification can be used only by that People activity. For details, see Creating an Inline Task or Notification.

- A local notification declared within either the scope containing the People activity or the process scope. In this case, the notification can be reused as an implementation of multiple People activities enclosed within the scope containing the notification declaration. For details, see Adding a Task or Notification to the Outline View for Process or Scope Use.

- An escalation action for a task deadline. For details, see About Task Deadlines and Escalations

- (For future release.) A standalone notification identified using a QName. In this case the notification can be reused across multiple BPEL4People processes within the same environment.

A notification definition contains the following parts:

- Notification Name, Priority, and Interface

- Role Assignments for a Task or Notification

- Adding Task or Notification Presentation Properties

- Adding Rendering Details for a Task or Notification

Required and Optional Properties for a Notification

The following table describes properties required to make a notification valid as well as properties you can optionally define:

Required Properties	Optional Properties
Interface	Description
Priority	Renderings
People Assignments	
Subject	

Notification Name Priority and Interface

Enter the port type and operation from the WSDL you want to use for the notification interface.

In the Properties view of a notification, the Notification tab displays the following properties:

- **Name**. (Required). The name combined with the target namespace of a notification element is used to uniquely identify the notification definition. Note that this is not the name displayed in users' task area in Process Central. That name is Display Name, as described in Adding Task or Notification Presentation Properties.

- **Priority** (Optional). You can set a default priority level as a non-negative integer, starting at zero as the highest priority. You can create the priority as an expression. The priority is displayed in users' Task Details view. Note that priority zero (0) is displayed with red flag. For an illustration of how a priority appears in a user's Task Details view, see Adding Task or Notification Presentation Properties.

- **Interface**. (Required). Specifies the WSDL port type and one-way operation that is the interface to the notification. The WSDL must be available in the Project Explorer.

If your process already contains an import for the WSDL that defines the Port Type and Operation then select the appropriate values.

Using the All Tab of a Notification

In a notification's Properties view, the All tab displays all available properties, including some that do not appear on the Notification, Presentation, and Assignment tabs.

You can define properties on the All tab or on the other tabs as desired.

For details on properties that appear only on the All tab, see Adding Rendering Details for a Task or Notification.

Using the People Activity

This chapter begins by discussing people activities. While these topics mention Guide Designer, you will need to read *Using Guide Designer User Tasks* for information. However, many of the following topics also relate to Guide Designer user tasks.

What is a People Activity

A people activity contains the instructions and data a person needs in order to complete a task. When a business process requires a decision from a person, rather than an invoked service, use a people activity. Within the Process Developer, this is represented as a User Task:
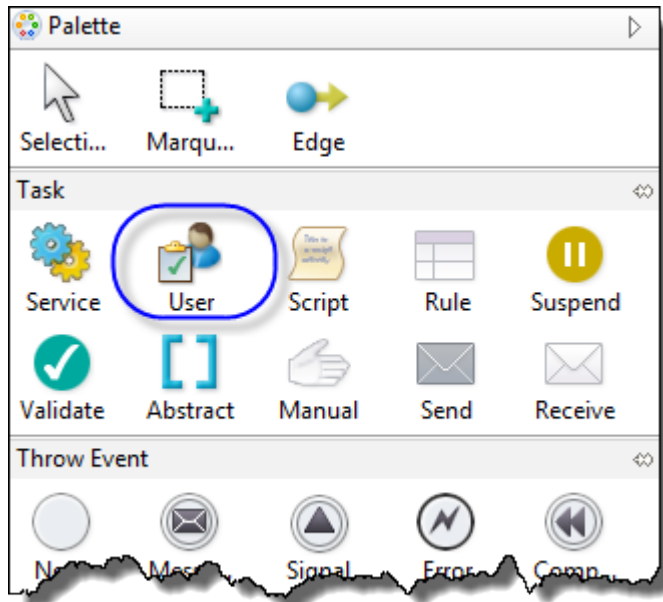
In BPMN, a people activity is implemented as one of the following, depending on whether the activity contains a task or notification:

- People Activity Containing a Task:
  - User Task
  - Manual Task
- People Activity Containing a Notification:
  - Send Task
  - Message Send Event

The People activity starts when a task is sent to potential task owners and is complete when the task's owner returns data to the process. Once the process receives the data, it can continue to the next execution step.

The following illustration shows a basic People activity, using a custom name, within a BPEL process.
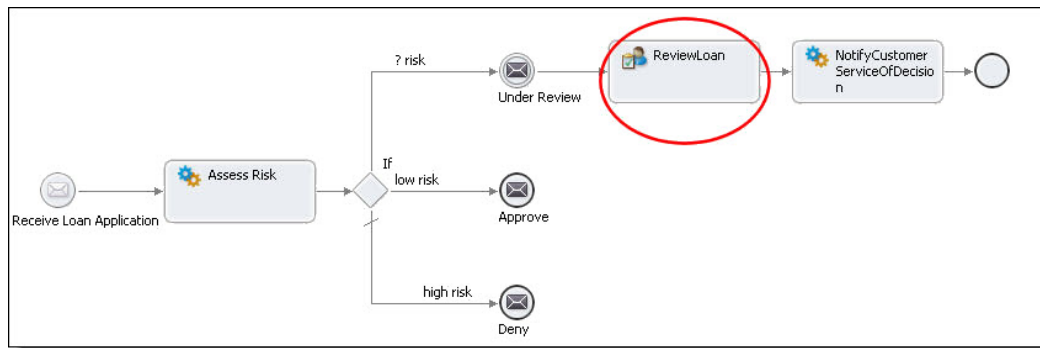
The people activity is built with a task definition and other details through the activity's Properties view or it can be built using Guide Designer. (See *Using Guide Designer User Tasks* for more information.)

When a BPEL process is deployed to the server and is instantiated, the task is sent to all potential owners and administrators. Anyone of these can claim ownership, work on, and complete the task.

For more details, see *Conceptual Overview of the People Activity*.

Conceptual Overview of the People Activity

A People activity is an extension activity that is a construct from the BPEL4-People specification. It contains a task or notification construct from the Human Task (WS-HT) specification.

A People activity can have one task or notification (plus overrides) and uses process variables to send data to the task or notification. It receives data into a process variable from a task. In addition to using a task or notification, a People activity has its own properties, including scheduled actions, attachments, and other attributes. If you are using Guide Designer, these are defined within the guide.

The following illustration shows the building blocks for a people activity:
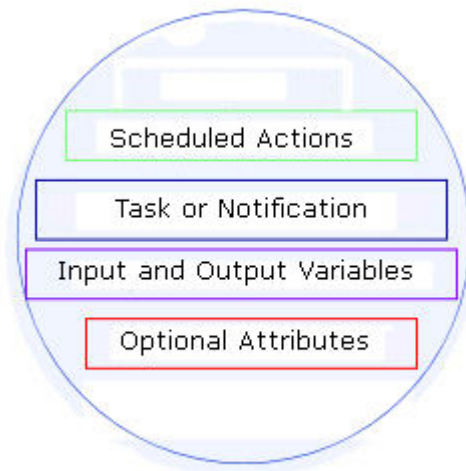
A People activity includes a task or notification with additional built-in workflow actions. The building blocks are divided into several properties including:

- **Scheduled Actions.** You can define a deferred time to start the activity and define an expiration time for the activity.
- **Task or Notification.** A People activity uses either a local or inline activity. For details on local activities, see *Human Interactions Extension Element*.
- **Input and Output Variables.** The task's interface, and optional call back interface, are used to provide the input and output data for the task. The notification's interface provides input data.
- **Optional Assignment and Priority Overrides.** For a People activity using a local task or notification, you can define new people assignments and priority level.

See also *Adding a People Activity*.

Adding a People Activity

Add a people activity in one of the following ways from the BPMN palette:

- To Create a People Activity Containing a Task, select:
    - User Task
    - Manual Task
- To Create a People Activity Containing a Notification, select:
    - Send Task
    - Message Send Event

Alternately, use the Participants view to automate some of the requirements for a task-based People activity.

Adding a People Activity from the BPMN Palette

Add a People activity to the process or a scope from the BPMN palette, as shown in the example. The activities in red squares are for task-type People activities and the ones in blue circles are for notification types.
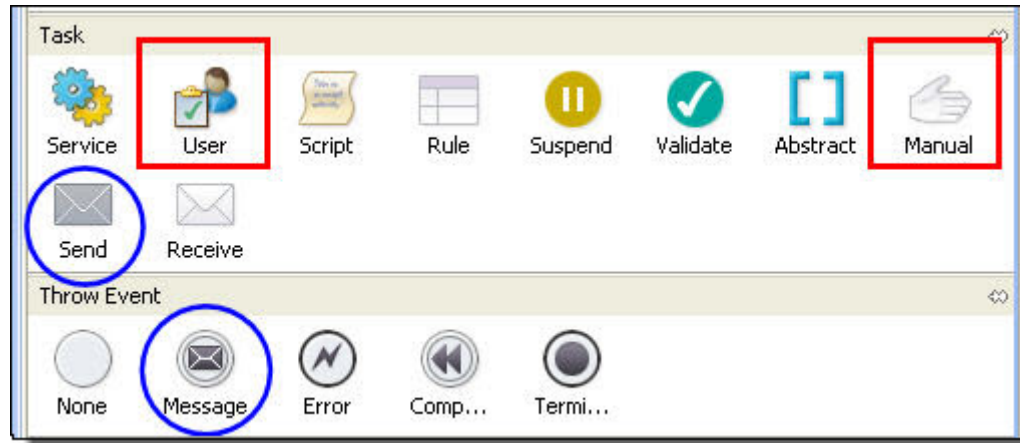
Fill in the required properties from the People Activity's Properties view. Note that you must create an inline task (or notification) or select a local task (or notification). For details, see *Required and Optional Properties of a People Activity*.

You may create either a People Activity of Guide Designer User task:
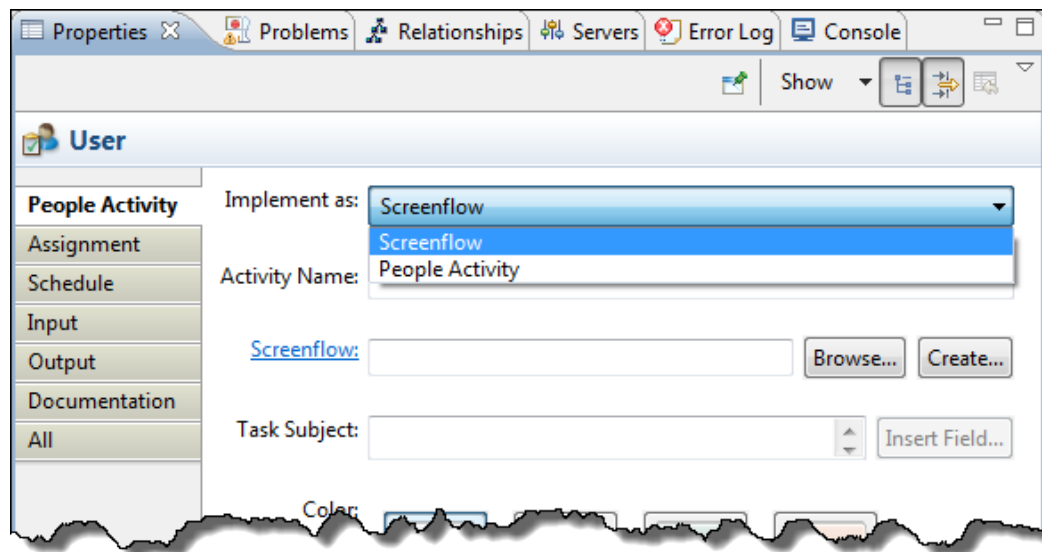
The left tabs are the same for both kinds of activities. However, the contents of the People Activity and Assignment tabs differ slightly. If you choose Guide Designer, the Input and Output tabs may have values set when you associate a guide with this user task. (See *Using Guide Designer User Tasks* for more information.)

Adding a People Activity Using the Participants View

You can use a shortcut for creating a People activity defined with a local task. The shortcut involves using the Participants view to create a Logical People Group and a local task for the group.

1. Create a new Human Task Participant and new Human Task.

2. From the Participants view, drag a human task to the Process Editor canvas.
   Notice that the People activity is created using the task interface and assignments definition.

3. In the Input tab, do one of the following:

   - Select Single Variable from the **Assignment Type** and select a variable.

   - Select an **XPaths** or XQuery**.** For details, see the *Process Developer Designer Online Help* topics, *Input Variable* and *From Part to Variable*.

4. Optionally, select an *output variable* or a *to part*. For details, see the *Process Developer Designer Online Help* topics, *Output Variable* and *From Variable to Part*.

5. Select other optional properties as desired.

Adding a People Activity Using the Create Activity Wizard

Use the Create Activity wizard to create a People activity with a local task. You must use a two-way operation for a task in this wizard. If you have a one-way operation for the task, create a People activity manually. Then add the interface and required callback.

1. Display the Interfaces View. It is not shown by default.

2. From the Interfaces Port Types elements, expand a Port Type, and drag an operation to the Process Editor.
   If you select from Partner Link Type, the wizard assumes you do not want to create a People Activity, since the People Activity does not require a partner link type. The Create Activity wizard appears.
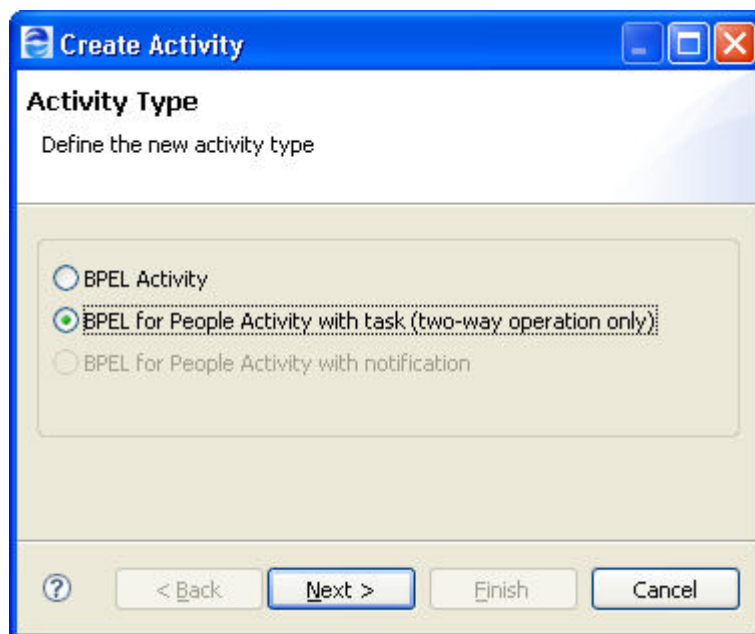
3. Select a task, if the operation contains both input and output messages, or a notification for a one-way operation, as described.
   If you want to create a task with a one-way operation and a callback, create the task manually.

4. Type in a task name.

5. Select the input and output variable names.

When you finish the wizard, a new People activity is created and a new local task appears within the Human Interactions element on the Outline view.

Add values for the properties of both the task and the People activity.

Adding a People Activity by Dragging a Guide

If guides already exist, you an create a people activity by dragging the guide onto the Process Editor canvas.
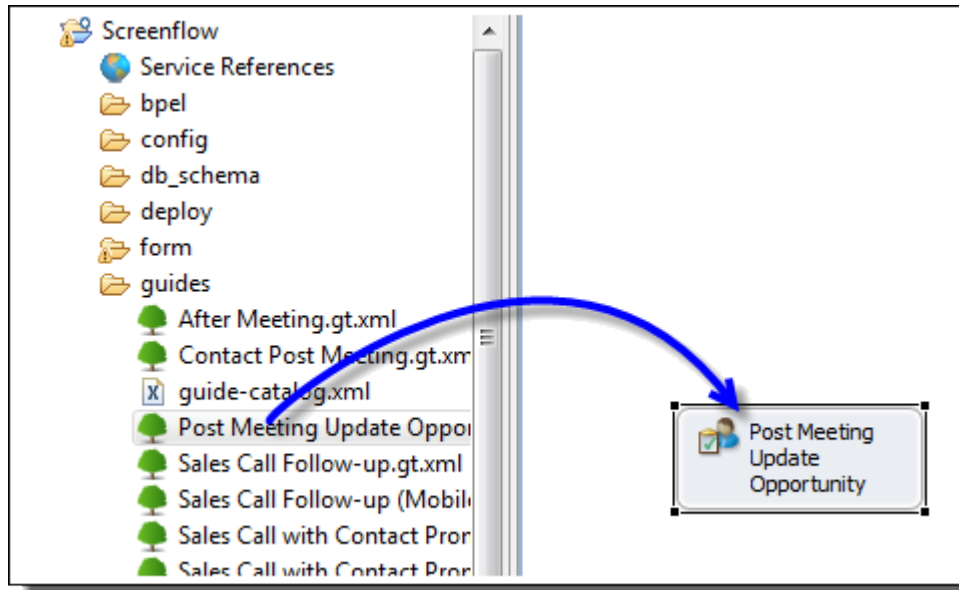
For more information, see *Using Guide Designer User Tasks*.

Required and Optional Properties of a People Activity

The following table shows the required and optional properties of a people activity.

Required Properties	Optional Properties
One of the following is required: - Inline task - Inline notification - Local task - Local notification For details, see *Creating an Inline Task or Notification* and *Selecting a Local Task or Notification*.	*Scheduled Actions*
Input variable or *toPart*. For details see, *Selecting Variables*.	Is skippable. See the topic below this table.
Output variable or *fromPart* (Only if task is selected. Not applicable for a notification.). For details, see *Selecting Variables*.	*People assignment and priority overrides*
	Attachments

Using the Is Skippable Attribute

This attribute indicates whether the task associated with the activity can be skipped at runtime. A person working on a human task or a business administrator may skip a task. A skipped task becomes obsolete, and no result is returned to the process. This attribute is optional. The default for this attribute is "no".

This attribute does not apply to a notification.

Creating an Inline Task or Notification

You can create a new, inline, task or notification for use only by the people activity. Alternately, you can select a task or notification created at the process or enclosing scope level, as described in *Selecting a Local Task or Notification*.
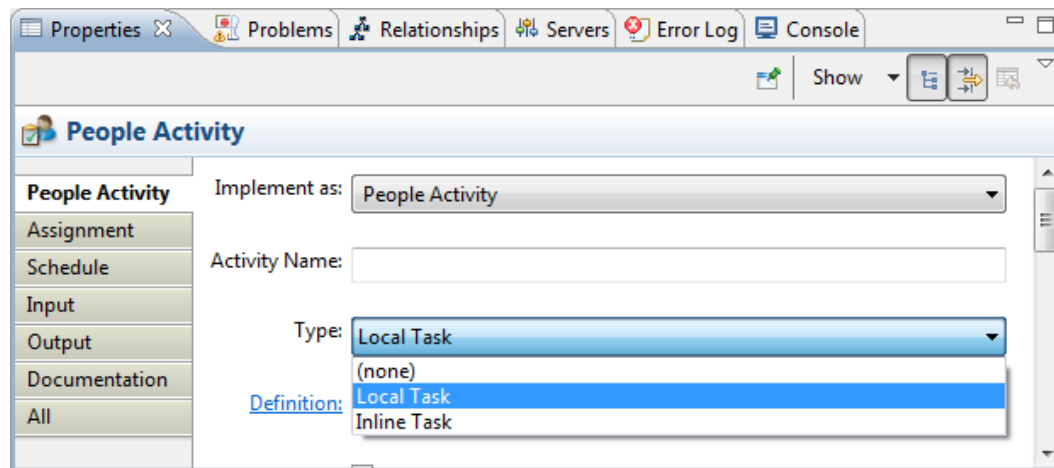
A people activity uses a task or notification definition, as described in See *Conceptual Overview of the People Activity*.

To Create an Inline Task or Notification:

1. From the Activity palette, drag the activity to the Process Editor.
   People Activity Containing a Task:

   - User Task

   - Manual Task

   People Activity Containing a Notification:

   - Send Task

   - Message Send Event

2. Select the activity to put the Properties view in focus.

3. Implement the activity as a people activity.

4. On the people activity main tab, select either Inline Task or Inline Notification from the Type picklist. A new task named T1 (or new notification named N1) is added to the people activity.

5. Select the **Definition** link. The Properties view of the new task is displayed.

6. At the end of the Task Interface field within the Task tab, select the Dialog (**...**) button.

7. Fill in the Interface details for the task.

8. In the Task or Notification Properties view, add values for the required properties.

Note: Be sure to define all required properties for the new task or notification. For details, see *Required and Optional Properties for a Task* and *Required and Optional Properties for a Notification*.

Selecting a Local Task or Notification

A people activity uses a task or notification definition. You can select a local task or notification that has been declared at the process or enclosing scope level. Alternately, you can create a task or notification inline for use only by one people activity, as described in *Creating an Inline Task or Notification*.

Note: Be sure to create local tasks and notifications first, in order to select from them in a people activity. For details, see *Adding a Task or Notification to the Outline View for Process or Scope Use*.

To Select a Local Task or Notification:

1. Add a new people activity as described in *Adding a People Activity*.

2. In the Properties view, from the Type picklist, select Local Task or, if appropriate, Local Notification. (These selections will not appear in the same drop-down list.)

3. From the Definition picklist, select the name of the task or notification that you want to use or select New Task or (if appropriate) New Notification.

If you want to override the people assignments and priority level specified in the local task or notification, see *Selecting Overrides for Priority and People Assignments*.

Selecting Overrides for Priority and People Assignments
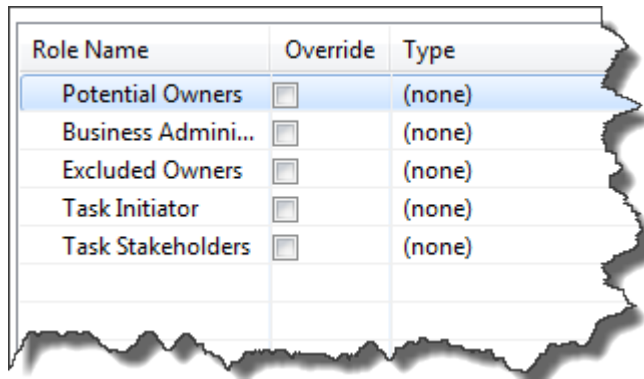
You can create a task or notification definition at the process or scope level and then use the task or notification in many People activities. Within each people activity, you can modify the priority and people assignments that were originally defined.

To Override Original Assignments and Priority:
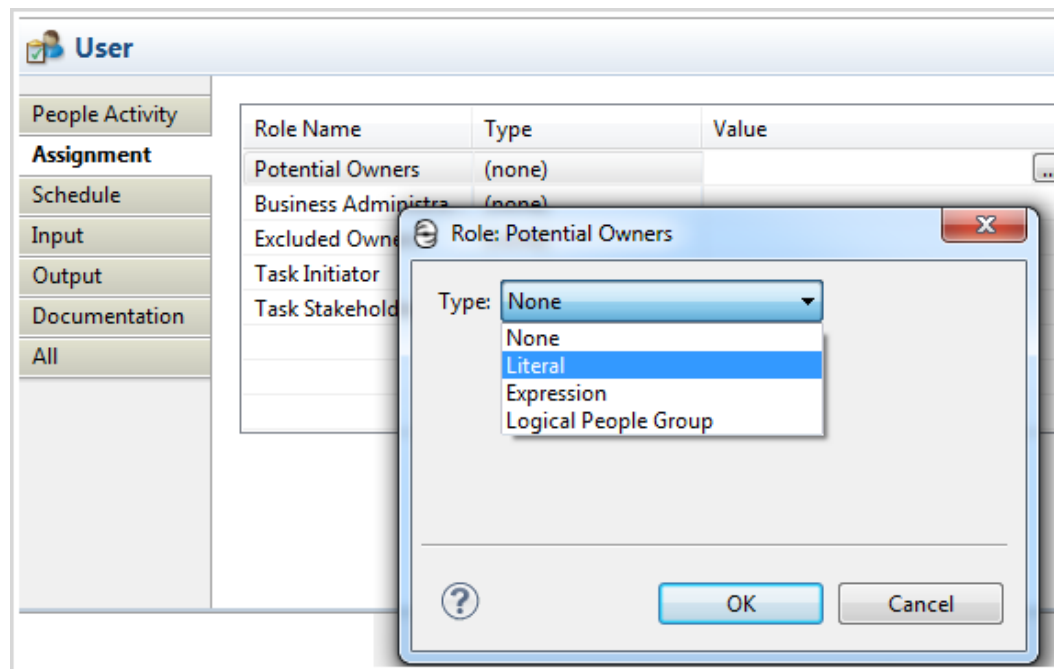
1. Select a people activity to put the Properties view in focus.

2. On the main People Activity tab, create a new expression for the priority.

3. On the Assignment tab, create new assignments for this activity.

The Assignment tab differs slightly between a People Activity and a Guide Designer.

The above snapshot has an Override column. This column does not appear if you are creating a Guide Designer. It is not there for Guide Designer because you do not need to override or go to the associated task.

Selecting Variables

In a people activity, you select or map to an input and an output process variable associated with a task's interface. The input variable contains the data displayed in a user's task client application. For a task, the data is what a user needs in order to complete the task. For a notification, it is the data that informs a user. The input variable copies data from the BPEL process into the task or notification input parts.

Rather than creating a variable, you can map variable parts using XPaths or XQuery. For details, see the *Process Developer Designer Online Help* topics, *Input Variable* and *From Part to Variable*.

The output variable, available only for a task, contains the data the user sends back to the process. The variable copies data to the BPEL process from the task's output parts, which represents the task's result data.

For details, see the *Process Developer Designer Online Help* topics, *Output Variable* and *From Variable to Part*.

Adding Scheduled Actions for Tasks

You can schedule activation and expiration times for tasks. These properties do not apply to notifications.
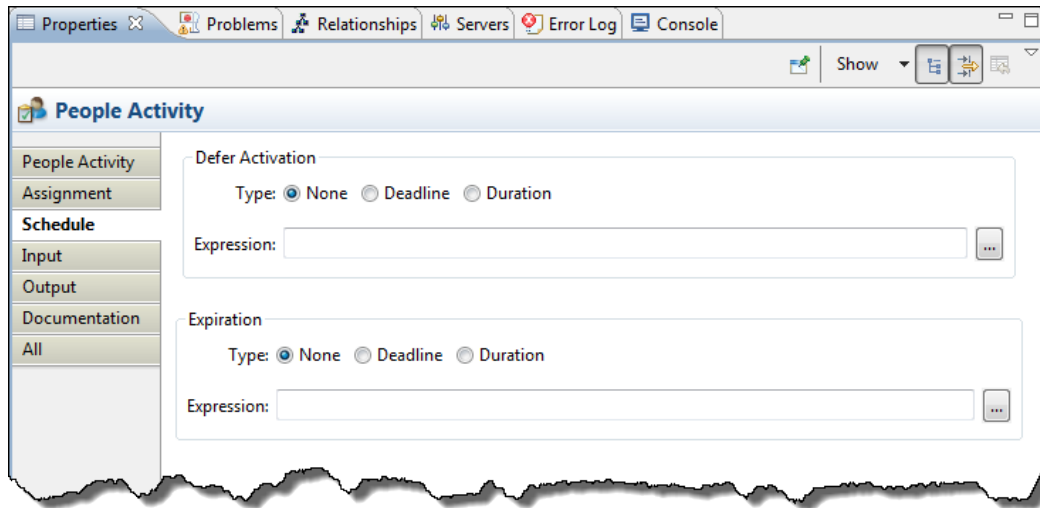
- **Deferred Activation for a Task**
  You can specify an alarm for when a task should become active. The alarm is defined as either the period of time after which the task reaches the Ready state (in case of explicit claim) or Reserved state (in case of implicit claim), or the point in time when the task reaches Ready or Reserved.

  For example, you can specify, "Activation of this task is deferred until the time specified in its input data."

- **Task Expiration**
  You can add a task expiration duration or date, if desired. The expiration indicates the date/time a task becomes obsolete, and the expiration date/time is displayed in users' task client application. When the expiration occurs, all task instances are listed as exited, and a fault is thrown in the BPEL process. A user cannot complete the task, and the output data, if sent back by a user, will not be used.

For example, you can specify, "This task expires when not completed within 14 days after having been activated".

To Specify a Deferred Activation:

1. Select the Properties view of a people activity.

2. Put the All tab in focus.

3. In the Defer Activation Alarm row, select Deadline or Duration.

4. In the Defer Activation Expression row, select the Dialog (**...**) button to open the Expression Builder.

5. Create either a deadline or duration expression. For examples, see "Deadline and Duration Expressions" in the *Process Developer Designer Online Help*.

Repeat the steps above to create an Expiration Alarm and Expiration Expression.

Sending and Receiving Attachments

An attachment is data that is relevant to a task but is not defined as one of the input or output parts. This data can be of any type. For example, you can attach a picture, document, spreadsheet, or other file type to a variable in a BPEL process. The people activity can receive attachments from the BPEL process. If attachments exist, the people activity can then make them part of the task, and the attachments are displayed in users' task client application.

Users can download and update the original attachments, or they can remove or add new attachments when they complete the task. When the people activity completes, it can copy all, none, or new attachments returned with the task to another process variable. If no attachments exist, the setting is ignored.

Using the _peopleActivityAttachments Variable to Copy Attachments

When you add a people activity to the process, a special variable is added to the Process Variables view for holding attachments.

For data attachments, you can create an assign activity with a copy operation that uses an attachment function to copy attachments from or to `_peopleActivityAttachments`. The attachment functions are listed in the Functions panel of the **Query Builder**, as shown in the illustration:
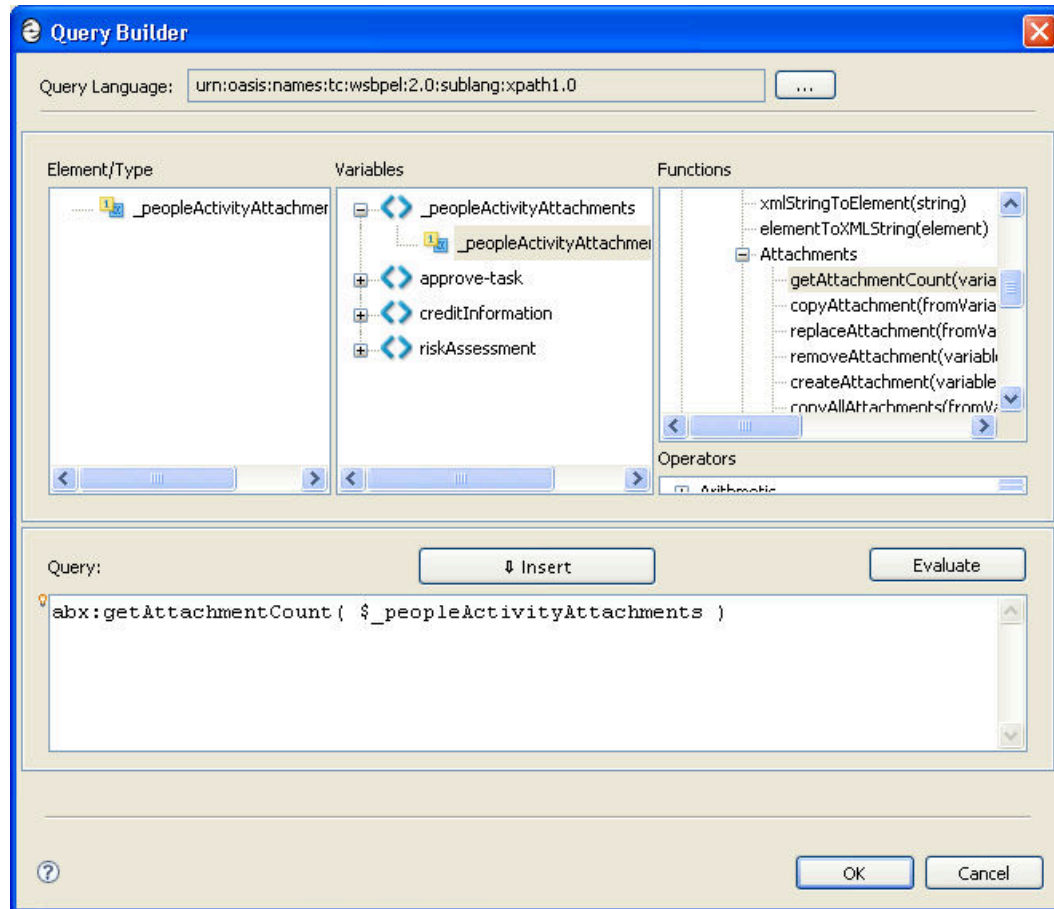
Setting the Attachment Propagation Property

In a people activity, on the All tab of the Properties view, you can make selections for copying attachments from the process to the people activity and from the people activity to the process:

- **From Process**. Select *All* or *None* to indicate whether to copy attachments to the `_peopleActivityAttachments`. The default is *All*.

- **To Process**. Select *All*, *None*, or *New Only* (but not those that were modified) to indicate which attachments to copy to a process variable from `_peopleActivityAttachments`. The default is *New Only*.

Catching a Fault Thrown by the People Activity

If an error occurs or a task expires, and the people activity cannot complete normally, the activity throws one of the following faults:

- `b4p:nonRecoverableError`
- `b4p:taskExpired`

If desired, you can catch one of these faults in a fault handler of an enclosing scope of a people activity.

Simulating Debugging Deploying

Simulation is an Process Developer feature, described in the Process Developer Help. This discussion focusses on all aspects.

Simulating a Process with a People Activity

Add sample values to the Logical People Group or other assignment role for testing.

Simulation is an Process Developer feature, described in more detail in the *Process Developer Guide*.

You can simulate the execution of a BPEL process that includes a People activity.

To Set Simulation Properties for a People activity:

1. Before you begin simulation, put the All tab of the People activity in focus.

2. Under the Simulation category, select values for fault and output messages and their attachments, as you normally do for a Web interaction activity, such as a reply.

3. Enter an optional value for an actual owner, if desired. See the information below on simulating with Logical People Groups.

4. Add comments, if desired, as described in *Simulating Comments Returned from a Task*.

5. Select an outcome from the Status list:

   - **Completed**. This is the task state for a normal completion.

   - **Failed**. Select this to simulate fault handling, and select the fault to simulate.

   - **Expired**. If you set a task expiration deadline, you can simulate the outcome. For fault handling details, see *Catching a Fault Thrown by the People Activity*

   - **Skipped**. If you selected the IsSkippable attribute for the task, you can simulate the process by skipping the People activity.

6. During or after simulation, you can view the data that is sent to the task client application. For details, see *Viewing Task Input Data While Simulating*.

To Simulate Logical People Groups:

You can select users or groups to simulate with if your task, assignment activity, transition condition, or other construct is using a Logical People Group.

1. On the Outline view, select a Logical People Group.

2. In the All tab, under the Simulation category, click (...).

3. In the **Organizational Entity Definition** dialog, select Generate User or Generate Group.

4. In the resulting literal definition, fill in the `<user>` or `<group>` elements with sample values.

Note: If you do not specify an Organizational Entity Definition, a default sample value of one group, named as the Logical People Group is automatically added as a simulation value.

To Simulate a Process Initiator:

If your People activity uses the B4P custom function `getProcessInitiator`, you can simulate the returned value by selecting a Process Initiator in the All tab of the process' Properties view.

Simulating Comments Returned from a Task

Generate comments for simulating comments that are returned by the task owner when the task is completed.

A task owner can add comments to the output data when returning the task to the process. You can simulate this behavior by adding simulated comments for the task output.

Also, note that you can use the getComments() function in your process to copy the comments from a People activity.

To Add Simulated Comments Returned From a Task:

1. Before you begin simulation, put the **All** tab of the People activity in focus.
2. Under the **Simulation** category, select **Task Simulation Comments**, and select the **Dialog** button in the *Value* column.
3. In the **Task Comments Definition** dialog, select **Generate Comments**.
4. Change or add values as desired, depending on how you intend to use the comments in the activities following the People activity.

Viewing Task Input Data While Simulating

View task data.

A People activity contains a large amount of data, much more than other activities. Given the number of expressions used in defining properties, such as task subject, description, deadlines, and escalation events, there is a special viewer provided to view task input data during simulation.

During or after simulation, this view shows how each expression used in the task definition is evaluated. For each task and People activity property defined by expression, you can see the evaluation.

To View Simulated Task Expression Results for Input to the Task:

1. Put the **All** tab of the People activity in focus. The People activity should have an associated task.
2. In the Execution category, select **Simulated Task Expressions**.
3. Select the Dialog (...) button to open the Simulated Task Expressions Result window.

The expression results include the following:

- Namespaces
- Input message sample data
- Presentation elements
- Assignments
- Deadlines
- Comments attached to the task input (but not comments attached to the output)

Selecting a Logical People Group Handler During Deployment

Add a new group in PDD for static assignment of a logical people group.

If your process defines a one or more Logical People Groups, you can use the PDD Editor to specify how to resolve the members of the group. The PDD Editor adds a People tab if it detects Logical People Groups.

To define logical people group handling:

1. Open the PDD Editor, as described in Creating a Process Deployment Descriptor File in *Process Developer Online Guide*.

2. On the People tab of the Editor, select the name of a Logical People Group.

3. From the Type list, select how to resolve the members of the Logical People Group:

    - **Use LPG Name as Identity Service Name**. For details see *Using or Mapping Users or Groups From the Identity Service*.

    - **Map to Identity Service** For details, see *Using or Mapping Users or Groups From the Identity Service*

    - **Dynamic**. You have assigned users or groups to a Logical People Group by using a copy operation in an assign activity. For details, see *Using a Logical People Group in an Assign Activity*.

    - **People Query**. For details, see *Adding a People Query*.

Using or Mapping Users or Groups From the Identity Service

If desired, you can name actual users or groups from your Identity Service to the PDD Logical People Group panel. Note that the WS-HT specification does not allow a mixture of users and groups.

If you selected a user or group name from the Identity Chooser when you created the Logical People Group, you can use that name in the PDD.

If your Logical People Group has a generic name, such as "Role1", you can map this name to an actual user or group in your Identity Service.

Before you can select actual groups, you must ensure that your Identity Service is correctly enabled, as described in *Setting up an Identity Service*.

To add individual users, select the Users radio button, and then select **Add**. In the **Add User** dialog box, type in the name of one user, named exactly the same as in the Identity Service. For example, if you are testing, you may have defined a user in the `tomcat-users.xml` file as:

```
<user name="testUser1" password="b4p" roles="abTaskClient,
Finance,Marketing,Accounting,NERegionReps" />
```

In this example, you can type in `testUser1` as a User in the PDD Editor.

To add Groups, select the Groups radio button, and then select **Add** to add a new group name. You can also select from an Identity Chooser, as described in *Using the Identity Chooser During Deployment*.

Adding a People Query

The most flexible way to specify actual users to work on tasks or to receive notifications is with a Logical People Group query. You can create a query that searches your enabled JDBC or LDAP Identity service for the users or groups that meet the search criteria.

Note: A People Query is not applicable for a file-based Identity service.

For example, you can create a query that performs the following search in a JDBC or LDAP Identity service: "Select the manager of the task owner to receive an email notification for an overdue task."

To Load the Identity Configuration:

The query you write is based on the users and groups and their attributes that are configured in your LDAP or JDBC Identity service. The Identity service is configured in the Process Console, and then you must set up communication with the server from within Designer, as described in *Setting up an Identity Service*.

After the set up is complete, select **Load Identity Configuration**. A message informs you if the configuration was successfully loaded. If the configuration is updated on the server, the updates are automatically loaded into Designer.

Select either the Users or Groups radio button, and then select **Query Builder** to create an expression for retrieving users.

To Use the People Query Builder:

When the People Query opens, you see the following items:

- The **Expression Language** is SPARQL, a language well-suited for retrieving identities from a data store. This is the required language for people queries.

- The **Attributes** column lists the Identity service's parameters that are currently available for queries. For example, users' attributes always include `userName` and `memberOf`, and may include many other attributes configured in Process Server.

- The **LPG Parameters** column lists any parameters defined for the currently selected Logical People Group. For details on adding LPG parameters, see *Creating a Logical People From the Outline View* and *Logical People Group Parameters and Arguments*.

- The **Query** text box contains the `FILTER()` keyword from a SPARQL query. The Filter clause is the only part of a SPARQL query that you need to complete. The fully qualified query is automatically generated by Process Developer.

To create a valid SPARQL query, you must complete the FILTER clause, as shown in the examples.

- `FILTER($userName="Jani Mani")`
  Select the user Jani Mani.

- `FILTER (($memberOf = "NE_Reps") || ($memberOf="NW_Reps"))`
  Select users from either NE Reps or NW Reps.

- `FILTER ( (($memberOf = "NE_Reps") || ($memberOf="NW_Reps")) && BOUND(?email))`
  Select users from either NE Reps or NW Reps who have email addresses.

- `FILTER ($countryCode = "${country_code}")`
  Select users in a certain country, where the Logical People Group parameter is `country_code` and the Identity Service search attribute is `countryCode`.

For details on writing SPARQL queries, see http://www.w3.org/TR/rdf-sparql-query/.

Using the Identity Chooser During Deployment

Use the Identity Chooser option to return a list of groups in your organization for selecting Logical People Groups during deployment.

Before selecting the Identity Chooser, you must do the following:

- Ensure that the Process Server is running

- Ensure that you have set up an Identity Service on the server, as described in *Setting up an Identity Service*.

- Ensure that the Process Developer Identity Service Preference page is filled in correctly. Select **Window > Preferences > Process Developer > Identity Service** to view and update this page.

To Access the Identity Chooser:

1. In the **People** tab of the PDD Editor, select Map to Identity Service User/Group Name(s) as the deployment type.

2. Select **Groups**.

3. Under Groups, select **Select** to open the Identity Chooser.

4. Select **Refresh** to view the current list of groups.
   A list of all the groups named in your Identity Service appears, as the example shows:
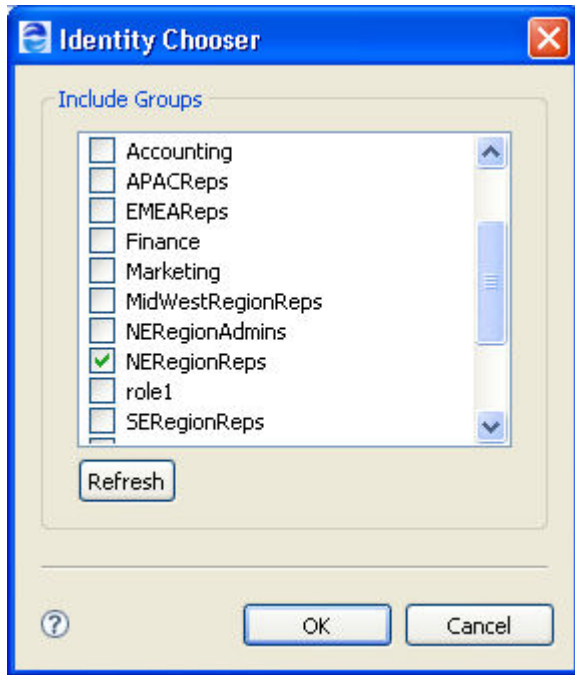
5. Select the checkbox next to each group that you want to include in your Logical People Group.

Running Your Process from the Process Server

There are a variety of Web services tools you can use to instantiate a BPEL process, including the Eclipse Web Tools Project that is included in the Process Developer installation.

Once your process is running, you can open Process Central by selecting the **Process Central** icon on the Process Developer toolbar. You can also point to the following URL in your Web browser:

```
http://[host:port]/activevos-central
```

where you point to the server where Process Central is deployed.

Be sure that the `abTaskClient` security role is mapped to each group member in the Identity Service who will login to Process Central.

You can sign into to work on the task routed there when the process executes the People activity.

For the older Process Central inbox application, use the following:

```
http://[host:port]/activevos-central/inbox
```

Setting Up Test Groups

For testing, you may want to set up a set of users or groups. Using your Process Server Identity Service, you can set up an XML file with user and group information that is helpful for testing the People activity. This means you do not have to set up all the details for connecting to your organization's LDAP or JDBC data store for testing. For details, refer to *Setting up an Identity Service*.

Providing Renderings for Task Clients

About Task Presentation in Process Central

Select a rendering.

Process Central is a client application that contains tasks and notifications, as well as Process Request Forms and Reports.

In Process Central, tasks and notifications are shown in a list, and users can select an item to work on and you can configure both the task list presentation as well as individual task presentation.

Here is an overview of the presentation properties you can configure.

Process Central Task List Presentation

All configuration of the task list is provided in a configuration file that you deploy. The basics include the following:

- Show or hide standard categories of tasks, such as open or closed tasks.
- Add your own folder names for tasks, based on different tasks in different People activities.
- Show or hide standard columns of a list item, such as Task Name, Subject, or Status.
- Add your own custom columns based on a task's Presentation Parameters. The columns are searchable. For details, see *Creating Custom Task Properties*.

Process Central Individual Task Presentation

Process Developer provides a comprehensive design environment for presenting a task to users. For details, see *Rendering a Task Interface into a User Interface*.

Rendering a Task Interface into a User Interface

Rendering is the application of a friendly user interface for the task's interface input and output messages. When a user views a task in a task client application, the details are displayed in an HTML form. A user reads the input message data and uses a variety of controls to enter and submit a response for the output message.

Informatica Business Process Manager has different rendering mechanisms for different task client applications.

- **Process Central**
  Generate HTML forms and edit them as desired. For details, see Creating an Process Central Task Form.
- **Customized client application**
  In addition to, or in place of the Informatica Business Process Manager renderings, you can provide completely customized client applications for sending tasks to telephones, computers, and other devices. For details, see Creating a Custom Rendering.

Creating an Process Central Task Form

Create an HTML form to render a task's input and output messages for ActiveVOS Central task client. The form opens in a Web Page Editor.

In the Renderings tab of a task's properties, you can create a new task form or browse to a file that is already available in the workspace.

Creating a Task Form

To Create a Task Form:

1. Display the Properties view for a task that you have defined.

2. In the Renderings tab of a Task's Properties view, select the Process Central Form tab, if needed.

3. Select **Create Form**.

4. In the **Create Form** dialog, select the default folder and filename for the form. The default filename is the task interface operation name.

5. Select **Advanced** to do the following:
   Select Merge input and output fields if you want to use the input fields to format the output. This selection is appropriate for some tasks, but not all. It is most useful for task forms in which the input data is very similar to the output. For example, if the task input is a work item, and the task output is the same work item that is being updated, you can generate much of the format for the output by merging input and output. For details, see Merging Input and Output Fields in a Task Form.

   Select the checkbox to Externalize Strings for multilingual support. For details, see Providing Multilingual Support for Task Forms.

6. Select **OK** to open the HTML form in the Web Page Editor.

See also:

- Working in Development Mode or Production Mode

- Understanding the Task Form Template

- Examples of Task Forms for End Users

- Adding a New Service Operation to a Task Form

- Adding Styles, Scripts, and Meta Data to `.avcconfig` for use in a task form in the *Process Developer Online Help*.

Merging Input and Output Fields in a Task Form

Create an HTML form to render a task's input and output messages for Process Central task client. The form opens in a Web Page Editor.

Use the Input to Output Merge Mappings dialog to map a task input message element to a task output element of the same type.

When a task form is generated, the form's input and output sections are generated as separate sections. In cases where the task's input and output messages have the same or similar elements, this leads to a lot of redundancy in the form that is generated. For example, in the case where the input and output messages are exactly the same, the input section provides a read-only view of the data, and the output section provides an editable view of the same fields. Also, initially, when the form is loaded, the output section would be empty and the task owner would have to reenter data from the input fields to the output fields. The use of merge mappings in these scenarios makes the form simpler and easier to use.

Merge mappings is a way to specify that certain sections of the input and output are the same, and are to be treated as such. When such a mapping is specified, the generated task form shows a consolidated view of those mapped elements. The fields contain initial values from the task's input data, and when the task form is saved, the values are saved to the task's output elements.

When you create a new task form, be sure to select the checkbox for Merge input and output fields. The task form wizard presents a dialog for you to add individual mappings between input fields and output fields.

For each element in input that needs to be mapped to an output element, create a new entry in the mappings table. Select **Add** to open the Merge Mapping dialog and select an input element and an output element. Validation messages in this dialog assist you in selecting compatible entries.

Working in Development Mode or Production Mode

Generate task details forms that are easy for Inbox users to use.

You can view and update the rendered task details layout in an Process Developer task client without redeploying the BPEL process or even creating a new instance of the process.

In the Renderings tab of the Task Properties view, you can enable Development Mode for testing your renderings. In this mode, when you deploy the process, the rendering files are read from the file system.

Rendering files must be on the same machine as Process Developer when in Development Mode.

Development Mode works as follows:

- You can edit the files, refresh the Process Developer task client, and see your results immediately.
- The file location is displayed as project-based. Behind the scenes, the files are treated as file system-based, readily available for editing.
- The rendering files are added to the deployment archive.
- The rendering files are not cached.

Note: If you do not enable Development Mode, you can logout and login to Process Central to view changes to your forms and configuration files.

When you enable the Development Mode, you can test your rendering files as follows:

1. Deploy the process, start it, and sign in to the Process Developer task client. Sign in as one of the users you specified in the people activity assignments.
2. In Designer, open a rendering file in an editor and make updates, as desired.
3. Refresh the task client to see your changes.

To Remove Development Mode**:**

When you are done editing the rendering files, you can do the following:

- Disable Development Mode by removing the check mark in the Task Renderings tab.
- In the Rendering tab, leave the file locations as project-based or edit the locations to make them Web-based.
- Redeploy the process via a BRPD file or by creating a new PDD and BPR.
- Deploy the rendering files as described in Deploying a Task Form and Properties Files.

Understanding the Task Form Template

During creation of a task form, Process Developer does the following:

- Creates an HTML document that is displayed in the Web Page Editor. The HTML document has several parts:
  - Task header
  - Task Request
  - Task Response
  - Additional Information
  - Save button
- Reads the schema of the task interface's input message and transforms the message definition into the Task Request area of the form. Each part of a message is rendered in a simple table. If the input message has two parts, then there are two tables that display.

- Reads the schema of the task interface's output message and transforms the message definition into the Task Response area. For details, see Examples of Task Forms for End Users.

- The **Additional Information** section is populated at runtime with details for the task subject, presentation, and priority as well as any comments or attachments that are added to the task.

- The **Save** button is added to the Task Response area allowing task owners to complete the task.

For details on modifying a task form, see "Editing HTML in Process Central Forms" in the *Process Designer* help.

Examples of Task Forms for End Users

As you generate task forms, you can view the HTML rendering.

Each part of a message is rendered in simple table, with each row occupying the HTML input field. For example, if the input message has two parts, then there are two tables that display.

Repeating elements are supported by repeating the UI pattern for each input element. Two command buttons are added: **Add** and **Remove** so that users can add or remove items such as parts for a purchase order.

Simple data types are rendered as HTML controls as follows:

Data Type/Attribute	HTML Control
boolean	radio button
numeric	text input
date	text input (includes a date chooser)
string	text input
xsi mixed	text input
xsi any	text input (User must enter well formed XML content to pass validation)
enumeration	selection box
maxOccurs="unbounded"	Add/Remove buttons
minOccurs="1"	asterisk added to the label to indicate "required field"

Adding a New Service Operation to a Task Form

If desired, you can modify an existing task form by adding or changing fields and functions generated from service operations. A service operation is a WSDL operation from a Workspace WSDL that represents the endpoint of a BPEL process running in an Process Server. The operation must have a service name associated with it.

For details, refer to *Adding a New Service Operation for a Request or Task Form* in the *Process Developer Online Help*.

Providing Multilingual Support for Task Forms

You can create task forms that support different languages. To do so, you create a new task form, plus a default properties file and additional properties files for each language supported. The steps to accomplish this support are described below.

To Create a New Task Form and a Default Properties File:

1. Create a new task form, described in Creating an Process Central Task Form.

2. In the **Create Form** dialog, select the Externalize Strings check box.

3. Select the dialog button next to Property File.

4. Name the new properties file. Do not include an underscore in the name.

5. Browse to the forms folder to store the file.

When you select to externalize strings, the following events occur:

- Schema elements in the form are generated by identifier rather than by name to allow for runtime replacement of strings found in a properties file

- A new link is added within the task form to indicate multilingual support:
  `<link rel="i18n" charset="UTF-8" type="text/plain" href="../myTaskForm.properties" />`

- The list of schema elements is added to the properties file as a key-value pair. The value of each key is the English wording.

- The properties file that you create is the default externalized strings file. It is for the en_US language.

For Each Additional Supported Language:

1. Make a copy of the default properties file and paste it into the project *form* folder.

2. Name the copied file with the syntax shown below. Note that you must use the exact same name for each file, followed by an underscore, followed by an ISO 639 Code for the Representation of Names of Languages.
   Syntax

   `myFile_[ISO639Code]_{optional locale code}.properties`

   Examples

   `myTaskForm_fr.properties`

   `myTaskForm_fr_CA.properties`

3. Open the file in a text editor and replace the English values with another language.

Note: You can also add multilingual support for a task's presentation elements, namely the Subject and Description. For details, see Adding MultiLingual Support for Task Subject, Description, and Display Name.

See also Deploying a Task Form and Properties Files.

Deploying a Task Form and Properties Files

A task form and multilingual properties file are resources that must be included in a deployment package.

All forms in the project forms folder are automatically added to a BPR export archive in the Additional Resources section.

Creating an Process Central Configuration File for Tasks

For details of this topic, see "Configuring Task Role Filters" in yjr *Process Developer Online Help*.

The Task Role Filters section of the `.avcconfig` file configures the following:

- New role names to add to the Roles list in the Task list or replace the basic names of User, Administrator, Initiator, and Stakeholder
- New items to add to the Show list to replace or add to Claimed and Unclaimed
- New categories to replace or add to the Task list names of Notifications, Open and Closed
- Users and groups allowed to view the new categories
- Custom columns for the task list; for details, see Creating Custom Task Properties.

Here is an illustration showing the Task categories you can customize:
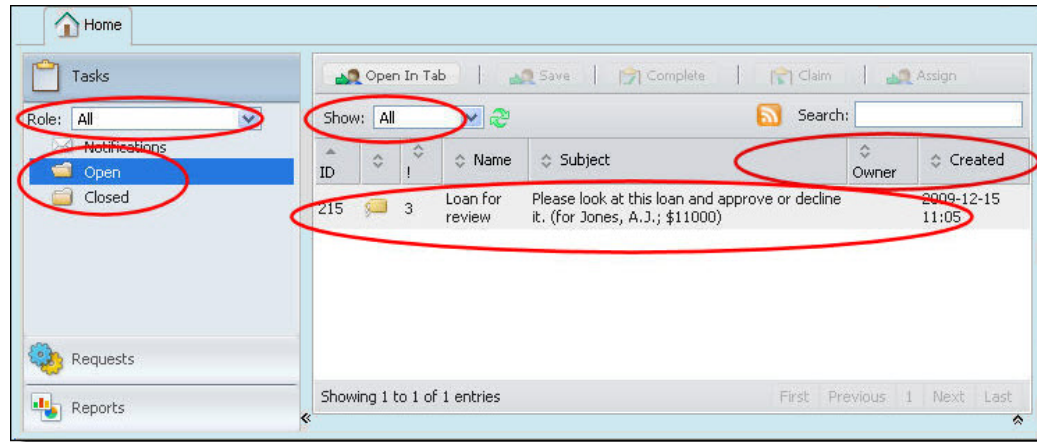
Creating a Custom Rendering

Type in the XML and other details for a custom rendering of a task or notification.

The Custom Rendering dialog is available when you want to provide your own custom client application for rendering task details. Use this dialog to provide XML rendering hints for your application.

In addition, you can add a custom rendering to the Renderings list. For details, see Contributing a Custom Task Rendering Hint Editor.

For an overview of this topic, see Rendering a Task Interface into a User Interface.

Contributing a Custom Task Rendering Hint Editor

Refer to the files to be used to provide rendering hints for this task or notification.

If desired, you can add a custom rendering to the rendering selection list. The rendering selection list includes custom. You can add to this list by contributing a custom rendering hint editor to Process Developer.

Your application can use the com.activee.bpel4peopleep plugin, which provides an extension point allowing you to contribute your own custom rendering hint editors. This extension point appears in the plugin.xml file.

You can contribute to the Task Rendering Hint Editor Extension Point.

The plugin.xml file already contains extension points for the JSP and XSL editors. You can use these as examples to contribute your own editor.

Custom Functions

Java custom functions cannot be used within a presentation parameter or in an expression editor when defining human tasks. This is not the same as the way in which you use other Java custom functions. If you wanted to use one, you would need to declare the function as being global in the Server Console; you could not refer to them in a contribution. While this works, it can be a problem as users would need to deploy and manage them independently from the package deployment process. Restated, unlike other functions, human task custom functions cannot be just included in a contribution so that a human task process can access it.

While using custom functions within a contribution appears to be a simple way of using them, this limitation prevents them from being easily used within a human task context.

This limitation exists because at runtime, the server expects the task state process to have references for custom function contributions, and this is not possible.

Human Tasks Custom Functions

b4p:getProcessStakeholders()

Description: Returns the stakeholders of the process.

b4p:getBusinessAdministrators()

Description: Returns the business administrators of the process.

b4p:getProcessInitiator()

Description: Returns the initiator of the process.

b4p:getLogicalPeopleGroup(Logical People Group Name)

Description: Returns the value of a logical people group

Parameter:

- LogicalPeopleGroupName: The name of the logical people group (xsd:string).

b4p:getActualOwner(People Activity Name)

Description: Returns the actual owner of the task associated with the people activity.

Parameter:

PeopleActivityName: The people activity name (xsd:string).

b4p:getTaskInitiator(People Activity Name)

Description: Returns the initiator of the task. Evaluates to an empty b4p:user in case there is no initiator.

Parameter:

- PeopleActivityName: The people activity name (xsd:string).

b4p:getTaskStakeholders(People Activity Name)

Description: Returns the stakeholders of the task. Evaluates to an empty b4p:organizationalEntity in case of an error.

Parameter:

- PeopleActivityName: The people activity name (xsd:string).

b4p:getPotentialOwners(People Activity Name)

Description: Returns the potential owners of the task associated with the people activity.

Parameter:

- PeopleActivityName: The people activity name (xsd:string).

b4p:getAdministrators(People Activity Name)

Description: Returns the administrators of the task associated with the people activity.

Parameter:

- PeopleActivityName: The people activity name (xsd:string).

b4p:getTaskPriority(People Activity Name)

Description: Returns the priority of the task associated with the people activity.

Parameter:

- PeopleActivityName: The people activity name (xsd:string).

abxb4p:getComments(People Activity Name)

Description: Returns the comments of given people activity.

Parameter:

- PeopleActivityName: The people activity name (xsd:string).

abxb4p:getTaskId([peopleActivityName])

Description: Returns the id of the WS-HT task given the people activity name.

Parameters:

- peopleActivityName: The people activity name (xsd:string).

abxb4p:getTaskUrl([peopleActivityName])

Description: Returns the URL to task service given the people activity name.

Parameters:

- peopleActivityName: The people activity name (xsd:string).

abxb4p:getInboxUrl()

Description: Returns the base URL to the task service application. Useful when sending email containing an Process Central link.

WS-HT (Human Task) Custom Functions

htd:getPotentialOwners([Task Name])

Description: Returns the potential owners of the task.Evaluates to an empty htd:organizationalEntity in case of an error. If the task name is not present the current task is considered.

Parameter:

- TaskName: The optional name of the Task to retrieve data from.

htd:getActualOwner([Task Name])

Description: Returns the actual owner of the task. Evaluates to an empty htd:user in case there is no actual owner. If the task name is not present the current task is considered.

Parameter:

- TaskName: The optional name of the Task to retrieve data from.

htd:getTaskInitiator([Task Name])

Description: Returns the initiator of the task. Evaluates to an empty htd:user in case there is no initiator. If the task name is not present the current task is considered.

Parameter:

- TaskName: The optional name of the Task to retrieve data from.

htd:getTaskStakeholders([Task Name])

Description: Returns the stakeholders of the task. Evaluates to an empty htd:organizationalEntity in case of an error. If the task name is not present the current task is considered.

Parameters:

- TaskName: The optional name of the Task to retrieve data from.

htd:getBusinessAdministrators([Task Name])

Description: Returns the business administrators of the task. Evaluates to an empty htd:organizationalEntity in case of an error. If the task name is not present the current task is considered.

Parameters:

- TaskName: The optional name of the Task to retrieve data from.

htd:getExcludedOwners([Task Name])

Description: Returns the excluded owners. Evaluates to an empty htd:organizationalEntity in case of an error. If the task name is not present the current task is considered.

Parameters:

- TaskName: The optional name of the Task to retrieve data from.

htd:getTaskPriority([Task Name])

Description: Returns the priority of the task. Evaluates to '-1' in case of an error. If the task name is not present the current task is considered.

Parameters:

- TaskName: The optional name of the Task to retrieve data from.

htd:getInput(Part Name, [Task Name])

Description: Returns the part of the task's input message. If the task name is not present the current task is considered.

Parameters:

- PartName: The name of the part within the task's input message.
- TaskName: The optional name of the Task to retrieve data from.

htd:getLogicalPeopleGroup(Logical People Group Name, [Task Name])

Description: Returns the value of a logical people group. In case of an error (e.g., when referencing a non existing logical people group) the htd:organizationalEntity contains an empty user list. If the task name is not present the current task is considered.

Parameters:

- LogicalPeopleGroupName: The name of an in-scope logical people group.
- TaskName: The optional name of the Task to retrieve data from.

htd:union(Set 1, Set 2)

Description: Constructs an organizationalEntity containing every user that occurs in either set1 or set2, eliminating duplicate users.

Parameters:

- Set1: The first set - one of htd:organizationalEntity, htd:users, or htd:user.
- Set2: The second set - one of htd:organizationalEntity, htd:users, or htd:user

htd:intersect(Set 1, Set 2)

Description: Constructs an organizationalEntity containing every user that occurs in both set1 or set2, eliminating duplicate users.

Parameters:

- Set1: The first set - one of htd:organizationalEntity, htd:users, or htd:user.
- Set2: The second set - one of htd:organizationalEntity, htd:users, or htd:user

htd:except(Set 1, Set 2)

Description: Constructs an organizationalEntity containing every user that occurs in set1 but not in set2.

Parameters:

- Set1: The first set - one of htd:organizationalEntity, htd:users, or htd:user.
- Set2: The second set - one of htd:organizationalEntity, htd:users, or htd:user.

Process Developer Extensions

abxht:getTaskId([taskName])

Description: Returns the id of the WS-HT task given the people activity name. If the task name is not present the current task is considered.

Parameter:

- TaskName: The optional name of the task to retrieve data from.

abxht:getTaskUrl([taskName])

Description: Returns the URL to task service. If the task name is not present the current task is considered.

Parameter:

- TaskName: The optional name of the task to retrieve data from.

abxht:getInboxUrl()

Description: Returns the base URL to the task service application.

abxht:getOutput([Part Name, [Task Name]])

Description: Returns the task's output message. If the part name is omitted, then the first part is used. If the task name is not present, then the current task is considered.

Parameters:

- PartName: The optional name of the part within the task's output message.
- TaskName: The optional name of the task to retrieve data from.

Using Customized Task Clients

Process Server implements the operations described by the OASIS WS-HumanTask (WS-HT) API task client specifications to manage human tasks as well as interact with specific tasks. In addition to WS-HT API, iy provides an extension API to the OASIS WS-HT API with enhanced functionality.

What is the Informatica Business Process Manager WS-HumanTask API

Process Server implements the operations described by the OASIS WS-HumanTask (WS-HT) API task client specifications to manage human tasks as well as interact with specific tasks. In addition to WS-HT API, Process Server provides an extension API to the OASIS WS-HT API with enhanced functionality.

You can download the Informatica Business Process Manager SDK from `http:/www.activevos.com`. This SDK contains the WS-HumanTask API to interact with tasks and to build a custom task client.

See also *Using the WS-HT and Identity Service SDK*.

Using the WS-HT and Identity Service SDK

The Informatica Web site InfoCenter contains the Process Server SDK, which includes an API for custom task client applications.

This SDK includes the documentation, JavaDoc, and samples for the use of the WS-Human Task and Identity Service.

Download the archive from the InfoCenter or Education Center.

Creating Custom Task Properties

A *task property* is a well-known WS-HT API property or a user-defined property that serves as a selection filter in Process Central and other task clients.

A task property can be used to display custom columns in Process Central's task view or other task clients. A task property can also be used to filter a task list or search within Process Central and other task clients.

- Using task properties, you can filter the Process Central task list like the following examples:
  - Indicate which tasks contain *comments* and *attachments*
  - Show all *auto* loan requests between *$4,000* and *$8,000*
  - Show all *mortgage* requests in *zip code* 06484
- The task properties required for the filters shown in the examples above consist of two types.
  - **WS-HT API properties**. The *comments* and *attachments* properties in Example One are standard task properties.
  - **User-defined properties**. The *loan type*, *loan amount*, and *zip code* properties in Examples Two and Three are custom properties created as Presentation Parameters in Process Developer.

To create a custom view for a task list, complete the following steps:

1. Decide which filters you want to apply to a task list.
2. Familiarize yourself with the WS-HT Task Property List.
3. Create your own properties, as described in *Creating a Custom Task Property*.
4. Create the task list configuration, described in *Configuring Process Central Task Columns and Task Filters Using Properties*.

WS-HT Task Property List

The following table shows well-known WS-HT properties. Note that the *Task* prefix is reserved to distinguish WS-HT task properties from user-defined properties.

WSHT Task Property Name (case-sensitive)	Type	Description
Task.ActivationTime	xsd:dateTime	Available if task activation was deferred
Task.CompleteBy	xsd:dateTime	Available if the task has a complete-by time
Task.CompleteByExists	xsd:boolean	
Task.CreatedOn	xsd:dateTime	The functions current-date(), current-time() and current-dateTime() can be used with date type columns. For example, Task.CreatedOn < current-date() filters all tasks created before today.
Task.Escalated	xsd:boolean	
Task.EscalatedOn	xsd:dateTime	Last escalated time (available if the task was escalated)
Task.ExpirationTime	xsd:dateTime	Available if the task has an expiration time. See comment for Task.CreatedOn.
Task.HasAttachments	xsd:boolean	
Task.HasComments	xsd:boolean	
Task.HasFault	xsd:boolean	
Task.HasOutput	xsd:boolean	
Task.ID	xsd.string	Task ID, such as urn:b4p:3.
Task.IsSkipable	xsd:boolean	Task is defined as "can be skipped"
Task.ModifiedOn	xsd:dateTime	Last modified time.
Task.Name	xsd.string	Local name of task (not the presentation name)
Task.Owner	xsd.string	Current owner of the task
Task.PaProcessId	xsd:long	Note that this is not a WS-HT property. It is an Informatica Business Process Manager property. The process ID of the BPEL process that contains the people activity
Task.PresName	xsd.string	Presentation name

WSHT Task Property Name (case-sensitive)	Type	Description
Task.PresSubject	xsd.string	Task presentation subject When using an Oracle database, you may get the following error: value too large for column "AEB4PTASK.SUMMARY". While it may appear that you have not exceeded the length limit, Oracle will be storing non-standard characters and these characters use more storage than the column can handle. You can fix this problem by setting the character set to AL16UTF16. However, you would need to take the database offline to make the change. as an alternative, you could use the following command: `ALTER TABLE AeB4pTask MODIFY Summary VARCHAR2 (254 char)`.
Task.PrimarySearchBy	xsd.string	Primary search by value
Task.Priority	xsd:integer	Task priority. Zero is the highest priority. Higher numbers are lower priorities.
Task.StartBy	xsd:dateTime	Available if the task has a start-by time
Task.StartByExists	xsd:boolean	
Task.Status	xsd.string	Task status such as CREATED, READY, RESERVED, IN_PROGRESS, COMPLETED, FAILED, ERROR, OBSOLETE
Task.TaskType	xsd.string	Identifies task or notification. The values are TASK or NOTIFICATION.

Creating a Custom Task Property

Create substitutable parameters for use in subject and description expressions. Also create custom task properties for selection filters in Process Central and other task clients.

When you define a task, you can create presentation parameters for the task subject and description, as described in *Configuring Presentation Parameters for a Task or Notification Subject or Description*.

You can also create presentation parameters that are used for custom task properties. By creating custom task properties, you can configure Process Central and other task clients to use them.

To use custom properties in Process Central, see *Configuring Process Central Task Columns and Task Filters Using Properties*.

To Create a Custom Task Property:

1. In the Properties view of a task, select the Presentation tab.

2. Click (**...**) to open the **Presentation Parameters** dialog.

3. Select Add and click (...) to open the Expression builder.

4. Create an expression for the parameter. For example, select an input message part. The name of this part can be used as a column name in an task list.

5. Select the Type. Allowed types are string, integer, double, boolean, date and dateTime

6. Name the parameter appropriately, as shown in the following illustration.
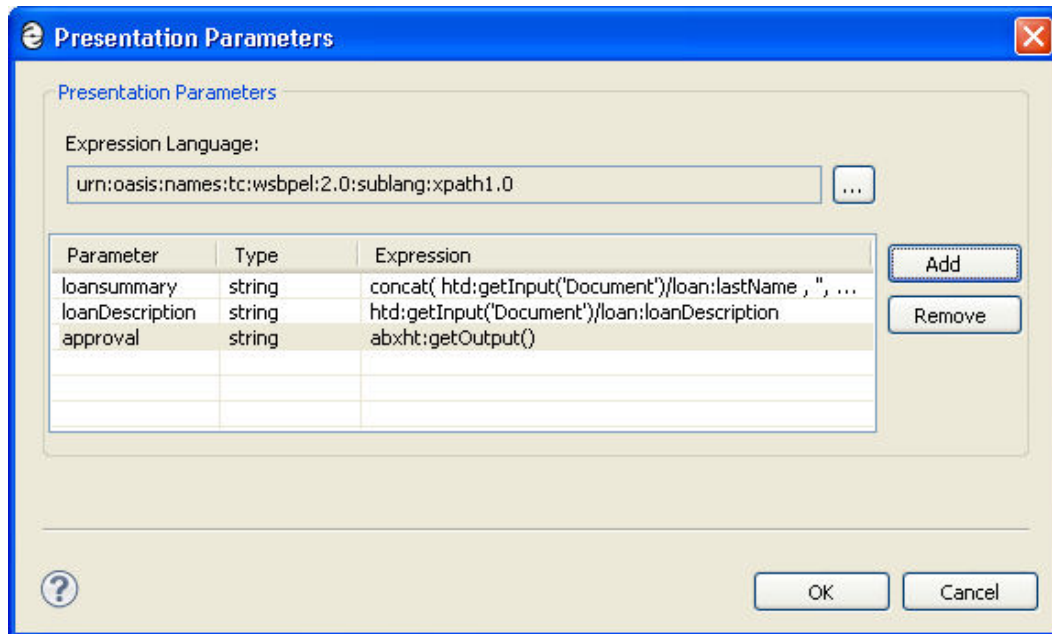
Tips for Using Custom Properties:

- Custom properties created in Presentation Parameters do not need to be used in the task subject or description.

- To view a list of the properties you create, deploy your process to the server, display the Process Console Catalog page, and select Task Properties

- Property names do not need to be unique, if added as different types. For example, you can create a property named "priority", once as a string and once as an integer. The Task Properties list appends the type to the name: *priority.string* and *priority.integer*.

- Allowed types are string, integer, double, boolean, date and dateTime.

- You may want to take advantage of the Process Developer custom function getOutput as an expression for a property. This allows you to filter a task list using an output message part.

- Task properties deployed in the .avcconfig file are not tied to a particular process. They are for general use in a task client.

See also, *Configuring Process Central Task Columns and Task Filters Using Properties*.

Configuring Process Central Task Columns and Task Filters Using Properties

For an overview of this topic, see *Creating Custom Task Properties*.

Create an Process Central configuration file by selecting **File > New > Central Configuration**.

For an overview, see *Creating an Process Central Configuration File for Tasks*.

Creating Custom Escalation Actions

In a task definition, you can specify one or more deadlines by which a task must be started and also one or more deadlines by which a task must be completed. If a particular start date or completion date is missed, you can trigger a particular action to occur.

There are several actions that are available, as described in *About Task Deadlines and Escalations*.

In addition, you can create and deploy a BPEL process to provide your own custom action.

To Create the Custom Notification Process:

1. Select **File > New > Bpel Process**.

2. Name the file and click next.

3. In the BPEL templates page, select Custom B4P Notification.

A process is started for you, with the receive already implemented. You can add the programming logic you wish.

The interface that is implemented is:

```
<portType name="IAeTaskCustomNotificationService">
 <operation name="processCustomNotification">
  <input message="aeb4pnw:processCustomNotificationRequest" />
 </operation>
</portType>
```

Adding the Service to the Start or Completion Deadline Property:

Once you have created and deployed your BPEL process, you can select process invocation, as described in *Invoking a Process When a Task Deadline Occurs*.

CHAPTER 36

BPEL Faults and Reports

This chapter includes the following topics:

BPEL Standard Faults

The table below specifies the standard faults defined within the WS-BPEL specification. All these faults are named within the WS-BPEL namespace with a standard prefix `bpel:` corresponding to the following URI:

http://docs.oasis-open.org/wsbpel/2.0/process/executable

Fault Name	Description
ambiguousReceive	Thrown when a business process instance simultaneously enables two or more IMAs for the same partnerLink, portType, or operation but different correlation sets, and the correlations of multiple of these activities match an incoming request message
completionConditionFailure	Thrown if upon completion of a directly enclosed `<scope>` activity within `<forEach>` activity, it can be determined that the completion condition can never be true
conflictingReceive	Thrown when more than one inbound message activity is enabled simultaneously for the same partner link, port type, operation, and correlation sets.
conflictingRequest	Thrown when more than one inbound message activity is open for the same partner link, operation, and message exchange
correlationViolation	Thrown when the contents of the messages that are processed in an `<invoke>`, `<receive>`, `<reply>`, `<onMessage>`, or `<onEvent>` do not match specified correlation information.
invalidBranchCondition	Thrown if the integer value used in the `<branches>` completion condition of `<forEach>` is larger than the number of directly enclosed `<scope>` activities.
invalidExpressionValue	Thrown when an expression used within a WS-BPEL construct (except `<assign>`) returns an invalid value with respect to the expected XML Schema type.
invalidVariables	Thrown when an XML Schema validation (implicit or explicit) of a variable value fails

Fault Name	Description
joinFailure	Thrown when the join condition of an activity evaluates to false and the value of the `suppresJoinFailure` attribute is **yes**
mismatchedAssignmentFailure	Thrown when incompatible types or incompatible XML infoset structure are encountered in an `<assign>` activity.
missingReply	Thrown when an inbound message activity executed, and the process instance or scope instance reaches the end of its execution without a corresponding `<reply>` activity being executed.
missingRequest	Thrown when a `<reply>` activity cannot be associated with an open inbound message activity by matching the partner link, operation, and message exchange tuple.
scopeInitializationFailure	Thrown if there is a problem creating any of the objects defined as part of scope initialization. This fault is always caught by the parent scope of the faulted scope.
selectionFailure	Thrown when a selection operation performed either in a function such as `bpel:getVariableProperty` or in an assignment, encounters an error.
subLanguageExecutionFault	Thrown when the execution of an expression results in an unhandled fault either in an expression language or query language.
uninitializedPartnerRole	Thrown when an `<invoke>` or `<assign>` activity references a partner link whose partnerRole endpoint reference is not initialized.
uninitializedVariable	Thrown when there is an attempt to access the value of an uninitialized variable, or in the case of a message type variable, one of its uninitialized parts.
unsupportedReference	Thrown when a WS-BPEL implementation fails to interpret the combination of the reference-scheme attribute and the content element or just the content element alone.
xsltInvalidSource	Thrown when the transformation source provided in a `bpel:doXslTransform` function call was not legal.
xsltStylesheetNotFound	Thrown when the named style sheet in a `bpel:doXslTransform` function call was not found.

User Reports Sample

This sample has reporting samples and information that will help you design and deploy your own reports. The first step is to create the report using the reporting module in Process Developer is built on the robust BIRT reporting engine. That engine has eclipse plug-in support, which Process Developer includes for you. These plugins contain a rich editing environment for the construction of new reports. Once you have created a report, you can then use a Business Process Archive (BPR file) to deploy it to the server as a resource.

After deployment, the report is available on the Report page of the console and optionally on the Reports tab of Process Central if you specify this using a Process Central configuration (`.avcconfig`) file. This sample's documentation supplies information about using reports with Process Developer, but only helps supplement

the reporting module's (BIRT) documentation. Please refer to the BIRT documentation for more details on advanced reporting features.

This topic discusses the following:

- Getting Started
- Sample Reports
- Deploying Reports
- Process Data Model
- Process Data Model Entity Relation Diagram Subset
- Human Task Data Model

Getting Started

You can add a new report to any project. It is good practice to separate the reporting files from other files in your project. In this example, the supplied reports are stored in the `report` folder. When you use **File > New** and choose to create a new report, always use the Process Developer Template. The template has a script, which when the report is deployed, automatically detects the Data Source connection information by using the supplied `AeBirtContext` class.

If you are interested in this script, open a report that uses the template, select the Data Source, and then click on the *Script* tab in the report editor. The Data Source while in Process Developer does not use the server settings; instead, it is predefined to point to your embedded Process Server's server derby database. This database has a limitation that only one application can connect to it at a time, so if you are creating reports using derby make sure you stop the embedded Process Server first.

Note: If you need to change the default, edit the data source properties to point to the database in which you store your data.

Once you have created a new report, the next step is to define a data set. A data set is the SQL and parameters that return the results from the database. After creating the SELECT statement, you can simply drag the data set onto the canvas and a simple table is created for you. Now you can go ahead and make any of the formatting changes for your report that you need.

Charts and other reporting types are also available. To use these features, you will need to consult the BIRT documentation. A few chart samples are included to give you a sense of the capabilities. Please note that these reports were designed and tested against the embedded server's derby database. You may need to make modifications to the SQL Data Set for each report if you want to use them with another supported database.

Sample Reports

Report	Description
IndexedPropertyValues	Displays all the indexed values in the system grouped by property name. Additionally it can show an individual property's values using the filter.
ProcessDistribution	Displays a pie chart of the distribution of process executions by process deployment (plan l, process name). Additionally, it has a listing of the counts.
OpenTaskRoles	Displays the users and groups and their roles for open tasks.
ProcessResponseTimes	Displays a chart showing process response times.
ProcessStatesByDate	Displays a chart showing process states by date.

ProcessStatesByProcess	Displays a chart showing process states by process name.
ProcessList	Displays a list of running processes similar to the Console's Active Process List. Clicking on a Process ID link opens the process view in a new tab. **Note**: This sample also shows how to create links to the console process detail view page.

Deploying Reports

Reports are easily deployed simply by selecting them or the folder where they reside using the context menu to select **Export**. From within the displayed dialog, select the **Orchestration > Business Process Archive** export option. Select a location for your BPR file. Typically, you will also choose Web service deployment with the location of your server (the default is localhost on port 8080). Also, make sure to check the *Replace existing resources* checkbox or the system may not overwrite the previous report deployments unless you have set that as the default when setting server properties.

You can also choose to save the deployment script, which is an ant script that can deploy your reports again simply by selecting **Execute** from the context menu. Note the sample project has a saved deployment script in the deploy folder that you can use to automatically deploy the sample reports. Selecting the **Next** button brings you to a page where you will see your reports in the additional resources list. Choose a *Process Group* name for both your entries, for example `Sample Reports`.

Process Data Model

Table 1. Table 1 - Process Developer Process Tables

Table	Description	Notes
AeUserProcessView(AeProcess)	Holds the main state of a process instance. The primary state of the process (for example, *running*, *started*, *completed*, and so on) is stored in the `ProcessState` column.	Actual BPEL language information and related services are stored in a related table, called `AePlan`.
AeVariable	This is the set of variable data associated with the process.	
AeIndexedPropertyValue	During deployment, you can choose to index fields from variables in your process. This table stores those values, which are typically your key performance indicators (KPI).	In order to find the property name associated with the value data, you need to link to `AeIndexedPropertyForPlan` (plan is the deployment information/table for a process definition) and the `AeIndexedProperty` tables.
AeProcessLogData	Contains the logging information associated with a process instance.	This is used to do root cause analysis (show process state at a point in time) of issues. It also contains useful reporting information.
AeProcessAttachment	Contains any attachments to variables in the process.	The actual attachment data is stored in the `AeAttachment` table as a BLOB.

AeAttachment	Stores the BLOB of an attachment.	
Others ...		

Process Data Model Entity Relation Diagram Subset

Table 2. Figure 1--ER Diagram for some of the process centric tables.
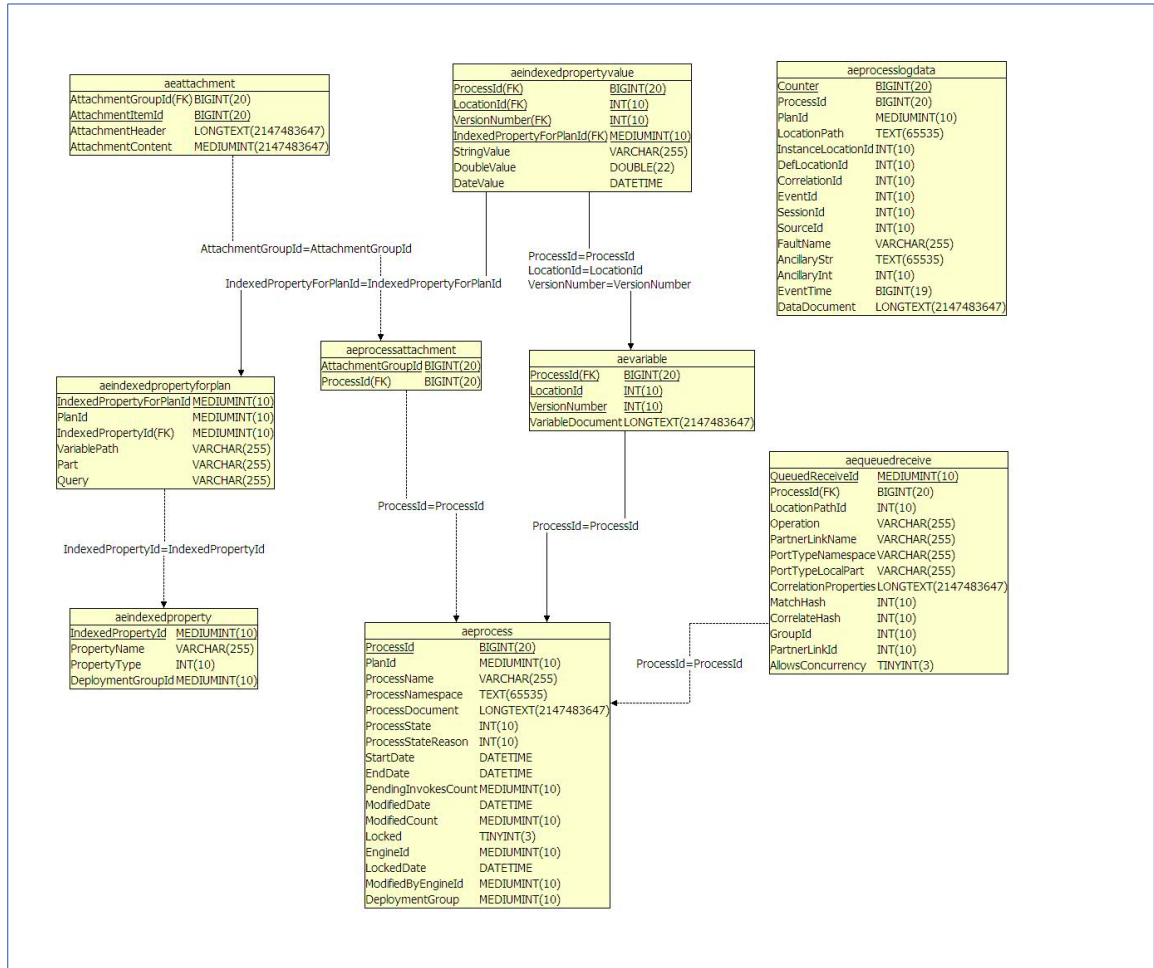
Human Task Data Model

In order to create human task-centric reports, you will need a basic understanding of the human task data model. The data model is of course centered on the task, which is in the `AeB4PTask` table. The primary key of this table is the `ProcessId`. Other tables have keys that relate back to the `ProcessId` in this table. As mentioned in the process section of the data model, a system process actually handles the task state management using the `AeB4PTask` table and the `AeProcess` table.

Table 2 contains a short list and descriptions of task tables for which you may have monitoring or other interests. Figure 2 contains a portion of the full data model that is centered on the `AeB4PTask` table.

Table 3. Table 2 - Human Task Tables

Table	Description	Notes
`AeB4PTask`	Holds the main state of a task instance. The primary state of the task (for example, *unclaimed*, *claimed*, *started*, *completed*, and so on) is stored in the State column see Table 4 below The `TaskType` column indicates whether the row contains a task or notification, (0=task, 1=notification).	Much of the task state interaction is accomplished through processes. Tochange anything in a task, you must use the WS-HT Service API. Most task access information is stored in `CLOB` columns in this table as organization entities. However, for quicker access when logging in, it is is also separated into its own table (`AeB4pTaskACL`).
`AeB4pTaskACL`	This table contains the access control list for the task. Type is zero for user and 1 for group. See Table 3 below for generic human role column values.	This table controls access to task information when using the WS-HT API. It controls what is visible to a user who is logged into the Process Central inbox, or another client application.
`AeB4PTaskPa AeB4PTaskEventDetail`	Contains all events that change the state of the task.	This is used to produce the history of a task.
`AeB4PTaskAttachments`	Contains the attachments associated with a task.	This is used to store attachments that are associated with a task and are typically available in the task detail display.
Others ...		

Table 4. Table 3--AeB4PTaskACL: GenericHumanRole Column

Role	Value	Description
Initiator	0	Task Initiator
Stakeholder	1	Task Stakeholder
Potential Owner	2	Potential Owner
Actual Owner	3	Actual Owner
Excluded Owner	4	An excluded owner
Business Administrator	5	Business Administrator
Notification recipient	6	Recipient of a notification task type

Table 5. Table 4--AeB4PTask: State Column

State	Value
Deferred	0

Unclaimed	1
Claimed	2
Started	3
Completed	4
Skipped	5
System Error	6
Faulted	7
Exited	8
Suspended	9

Table 6. Figure 2--Human Task ER Diagram Subset