



Using the BPEL Designer and Service Engine



Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 821-0017
June 2009

Copyright 2009 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun[™] Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Contents

Using the BPEL Designer and Service Engine	7
Overview	8
The JBI Runtime Environment	8
The BPEL Designer	9
The BPEL Service Engine	10
The Composite Application Project	11
BPEL Designer and Service Engine Features	11
BPEL Service Engine Features	11
Supported WS-BPEL 2.0 Constructs	12
Understanding the BPEL Module Project	23
Creating Sample Processes in the BPEL Designer	24
Navigating in the BPEL Designer	26
The BPEL Designer Window	26
The BPEL Editor Views	27
Element Documentation and Report Generation	28
The Navigator Window	29
The Properties Window	32
Scrolling	32
Collapsing and Expanding Process Blocks in the Diagram	32
Zooming In and Out of the Diagram	33
Printing BPEL Diagrams and Source Files	33
Creating a BPEL Module Project	35
Software Requirements and Installation	36
Starting GlassFish	36
Creating a new BPEL Module Project	38
Creating the XML Schema and the WSDL Document	39
Creating a BPEL Process Using the BPEL Designer	40
Creating a Composite Application Project	45

Building and Deploying the Composite Application Project	46
Testing the Composite Application	47
Summary	49
Developing a BPEL Process Using the Diagram	49
The BPEL Diagram	50
Configuring Element Properties in the Design View	51
Finding Usages of BPEL Components	51
Saving Your Changes	52
The BPEL Designer Palette Elements	52
Placeholders	53
The Process Element	54
The Web Service Elements	55
Using the Invoke Element	55
Using the Receive Element	57
Using the Reply Element	59
Using the Partner Link Element	61
The Basic Activities	66
Using the Assign Element	66
Using the JavaScript Element	67
Using the Validate Element	69
Using the Empty Element	70
Using the Wait Element	70
Using the Throw Element	71
Using the Rethrow Element	72
Using the Exit Element	72
Using the Compensate Element	72
Using the CompensateScope Element	73
The Structured Activities	74
Using the If Element	74
Using the While Element	75
Using the Repeat Until Element	76
Using the For Each Element	76
Using the Pick Element	77
Using the Flow Element	78
Using the Sequence Element	79
Using the Scope Element	80

Using the BPEL Mapper	82
About the BPEL Mapper	82
Creating BPEL Mappings	83
Working with Predicates	85
XPath Function Reference	86
Mapping Examples	89
Using Type Cast and Pseudo-Components	92
Using Normalized Message Properties	96
Using Normalized Message Properties in a BPEL Process	96
General Normalized Message Properties	101
Using Handlers	103
Using a Fault Handler	103
Using an Event Handler	105
Using a Compensation Handler	108
Using a Termination Handler	108
Using Correlation	109
Understanding Correlation. Using the Correlation Wizard	109
Validation	114
Validation Criteria	114
Validation Types	114
Notifications	115
BPEL Process Logging and Alerting	117
Defining Logging	118
Defining Alerting	121
Configuring the BPEL Service Engine Runtime Properties	122
Accessing the BPEL Service Engine Runtime Properties	122
Runtime Property Descriptions	123
BPEL Service Engine Deployment Artifacts	127
Testing and Debugging BPEL Processes	128
Testing a BPEL Process	128
Debugging BPEL Processes	131
BPEL Debugger Windows	139
Monitoring the BPEL Service Engine	146
Installing the BPEL Monitor API and Command Line Monitoring Tool	146
Using the BPEL Monitor Command Line Tool	146
Configuring Quality of Service (QOS) Properties, Throttling, and Redelivery	150

Configuring the Quality of Service Properties	150
Quality of Service Properties	152
Configuring Message Throttling	155
Configuring Redelivery	155
Using Dynamic Partner Links and Dynamic Addressing	157
Using a Literal to Construct an Endpoint	157
Using an Existing Partner Link's Endpoint	158
Using an Incoming Message to Extract the Endpoint	159
Using a Database Query to Provide an Endpoint	160
Sending Service Endpoint References	160
Configuring Persistence for the BPEL Service Engine	161
Setting the JVM Classpath to the Database JDBC Drivers	162
Configuring the User and Database for Persistence	163
Creating an XA Connection Pool and a JDBC Resource	165
Creating a Non-XA Connection Pool and JDBC Resource	167
Enabling Persistence for the BPEL Service Engine	167
Truncating and Dropping Tables	168
Configuring Failover for the BPEL Service Engine	169
Failover Considerations	169
BPEL BluePrints	170
Troubleshooting	170
Using BPEL Schemas Different from the BPEL 2.0 Specification	171
Ports	172
Travel Reservation Service Endpoint Conflict	173
Test Run	175
Test Run Failures	175
Disabling Firewalls when Using Servers	175
Required Correlation Set Usage is Not Detected by the Validation System	176

Using the BPEL Designer and Service Engine

One of the primary means of orchestrating web services is the use of Business Process Execution Language (BPEL). This guide provides an overview of the the BPEL Designer and the BPEL Service Engine, and describes how to use these tools to design, edit, compile, and deploy BPEL processes ways in which the IDE enables you to edit, compile, and deploy BPEL processes compliant with the WS-BPEL 2.0 specification.

The BPEL Designer is a graphic BPEL editor that enables you to easily create and edit BPEL processes, deploy them to the BPEL Service Engine, and run these processes in test or debug modes.

The BPEL Service Engine is a JSR 208-compliant JBI runtime component that provides services for executing WS-BPEL 2.0, an XML-based language used to program business processes.

What You Need to Know

The following topics contain introductory and conceptual information for the BPEL Designer and Service Engine.

- [“Overview” on page 8](#)
- [“BPEL Designer and Service Engine Features” on page 11](#)
- [“Understanding the BPEL Module Project” on page 23](#)
- [“Navigating in the BPEL Designer” on page 26](#)

Creating a Project

The following topics contain instructions for using the BPEL Designer to create a BPEL Module Project.

- [“Creating a BPEL Module Project” on page 35](#)

Using the BPEL Designer

The following topics contain information about using the BPEL Designer.

- [“Developing a BPEL Process Using the Diagram” on page 49](#)

- “The BPEL Designer Palette Elements” on page 52
- “Using the BPEL Mapper” on page 82
- “Using Normalized Message Properties” on page 96
- “Using Handlers” on page 103
- “Using Correlation” on page 109
- “Validation” on page 114
- “BPEL Process Logging and Alerting” on page 117

Using the BPEL Service Engine

The following topics contain information about using the BPEL Service Engine in a project.

- “Configuring the BPEL Service Engine Runtime Properties” on page 122
- “BPEL Service Engine Deployment Artifacts” on page 127
- “Testing and Debugging BPEL Processes” on page 128
- “Monitoring the BPEL Service Engine” on page 146

Advanced Operations

The following topics contain information about advanced operations for the BPEL Designer and Service Engine.

- “Configuring Quality of Service (QOS) Properties, Throttling, and Redelivery” on page 150
- “Using Dynamic Partner Links and Dynamic Addressing” on page 157
- “Configuring Persistence for the BPEL Service Engine” on page 161
- “Configuring Failover for the BPEL Service Engine” on page 169
- “BPEL BluePrints” on page 170
- “Troubleshooting” on page 170

Overview

The section covers the following topics:

“The JBI Runtime Environment” on page 8

“The BPEL Designer” on page 9

“The BPEL Service Engine” on page 10

“The Composite Application Project” on page 11

The JBI Runtime Environment

The Java Business Integration (JBI) runtime environment provides the runtime capability for SOA tools in the NetBeans™ IDE. The JBI runtime environment includes several components that interact using a services model. This model is based on Web Services Description Language

(WSDL) 2.0. Components that supply or consume services within the JBI environment are referred to as Service Engines. One of these components is the [“The BPEL Service Engine” on page 10](#) that provides services for executing business processes. Components that provide access to services that are external to the JBI environment are called Binding Components.

JBI components are installed as part of the GlassFish application server, which is packaged with the NetBeans IDE.

To view the installed or deployed JBI components:

1. In the IDE, open the Services window, expand the `GLASSFISH V2` node and expand the JBI node.
2. If you do not see the JBI node, you need to start the Application Server by choosing Start from the pop-up menu of the `GLASSFISH V2` node.



For a detailed overview of the Java Business Integration concept and a description of JBI nodes, see the [JBI Component Technical Overview](#).

The BPEL Designer

The BPEL Designer provides a highly-graphic framework that allows you to create and visualize business processes that are compliant with the WS-BPEL 2.0 specification. The BPEL Designer

feature of the NetBeans IDE allows you to easily create and edit BPEL processes. These processes can then be executed by the BPEL Service Engine on the GlassFish Application Server.

The BPEL Designer consists of four editing windows called views:

- **Source View:** The Source tab displays the underlying code for the business process. You can use the Source view to write your entire business process if you like, or just use it to review and edit the underlying code created when using the BPEL Designer's automated features to create your business process.
- **Design View:** The Design view provides a highly-graphical BPEL editor that lets you visually author a diagram of your business process by adding, editing, configuring and deleting BPEL elements. Elements are selected from the BPEL Designer Palette and dropped directly into the Design diagram. The constructed diagram in the Design view is automatically generated into BPEL source code compliant with the BPEL 2.0 specification.
- **Mapper View:** The Mapper tab provides a framework for processing and directing business process data. The Mapper is designed to enable you to graphically edit activities that have various expressions: assignments, conditions, and queries, the most common of these being XPath expressions.
- **Logging View:** The Logging mapper enables you to graphically define and "tune up" server side logging. Logging is used to write specified expression values or partner link endpoint reference information to the server log, and alerting enables the user to be notified of this information. Logging and alerting are supported for almost all BPEL activities.

The BPEL Service Engine

The BPEL Service Engine provides runtime services for deploying BPEL processes. The BPEL Service Engine is used to execute WS-BPEL 2.0 (or simply BPEL) compliant business processes. WS-BPEL 2.0 (Web Services Business Process Execution Language) is an XML-based language used to program business processes.

Business processes typically involve the exchange, or *orchestration*, of messages between the process and other web services known as *partner services*. The contract between a business process and partner services is described in WSDL 1.1. The message exchange between a business process and partner services is wrapped in the WSDL 1.1 message wrapper, as defined by the JBI specification, and routed via the JBI Normalized Message Router (NMR). The NMR interacts with external web services, not resident on the local JVM, via binding components. Binding components are responsible for encapsulating protocol-specific details. Transactions between the BPEL Service Engine and collocated EJBs or web components are handled through the Java EE service engine.

WS-BPEL 2.0 utilizes several XML specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0, and XSLT 1.0. Note that the JBI specification is targeted toward WSDL 2.0 and accommodates WSDL 1.1 by defining the wrapper. The BPEL Service Engine supports one-way, request-response operations (as defined in WSDL 1.1), within stateful, long-running

interactions that involve two or more parties. Asynchronous request-response is accomplished using two one-way operations, one implemented by a partner, the other implemented by the business process using correlation.

The Composite Application Project

The Composite Application project is used to create a Service Assembly that can be deployed to the Java Business Integration (JBI) runtime environment.

Within the Composite Application project, you can:

- Assemble an application that uses multiple project types (for example, BPEL Module or XSLT Module projects).
- Configure external/edge access protocols (SOAP, JMS, SMTP, and others).
- Build JBI deployment packages.
- Deploy the application image to the target JBI component.
- Monitor the status of JBI components and applications.

To deploy a Composite Application to the BPEL Service Engine, it must include a JBI module created from a BPEL Module project. Within a Composite Application Project that includes a JBI module, you can also create and execute test cases that can then be run against the deployed BPEL processes.

For more information about working with Composite Application projects, see [“Understanding the BPEL Module Project” on page 23](#) and [“Testing and Debugging BPEL Processes” on page 128](#) sections of this guide.

BPEL Designer and Service Engine Features

This section contains the following information:

- BPEL Service Engine Features
- BPEL Designer Features
- Supported WS-BPEL 2.0 Constructs

BPEL Service Engine Features

Following features are supported by the BPEL Service Engine:

- Standard JBI 1.0 engine component
- Supports BPEL 2.0 Specification
- Provides and consumes web services defined by using WSDL 1.1

- Exchanges messages in JBI-defined XML document format for wrapped WSDL 1.1 message parts
- Implements endpoint status monitoring
- Supports multiple-thread execution
- Supports debugging of business processes
- Supports database persistence of business process instances for reliable recovery from system failure
- Supports load balancing and failover when clustered

Supported WS-BPEL 2.0 Constructs

The following WS-BPEL 2.0 constructs are implemented by the BPEL Service Engine

Features	Support
<p>Process</p> <p>WS-BPEL 2.0 process root element. The Process element is present in the BPEL Designer diagram by default.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> extensions import partnerLinks variables correlationSets faultHandlers eventHandlers <p>Supported Attributes</p> <ul style="list-style-type: none"> name targetNamespace <p><i>Not Supported</i></p> <ul style="list-style-type: none"> queryLanguage (xpath only) expressionLanguage (xpath only) suppressJoinFailure abstractProcess exitOnStandardFault <p>Extensions</p> <p>More Information</p>

Features	Support
<p>Variable</p> <p>Supplies the mechanism used to hold messages that make up the state of a business process.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> name messageType type element <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Correlation</p> <p>Tracks the multiple long-running exchanges of messages that typically take place between a BPEL process and its partner services. The correlation mechanism helps to route messages to appropriate process instances.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>EventHandlers</p> <p>Invokes a specific action concurrently with a specified corresponding event.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> onEvent onAlarm <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>FaultHandlers</p> <p>Defines the activities that are executed as a response to faults resulting from invoked services.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> catch catchAll <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Import</p> <p>Used within a process to clearly express dependency upon external XML Schema or WSDL definitions. The Process element can have any number of Import elements as initial children, preceding any other child element.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <ul style="list-style-type: none"> namespace location importType <p>Extensions</p> <p>More Information</p>
<p>Web Services</p>	

Features	Support
<p>Invoke</p> <p>Invokes a one-way or request-response operation on a portType offered by a partner. It enables the business process to send messages to partners. The operation is defined in the partner's WSDL file.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> correlations toPart fromPart <p><i>Not Supported</i></p> <ul style="list-style-type: none"> catch catchAll compensationHandler <p>Supported Attributes</p> <ul style="list-style-type: none"> partnerLink portType operation inputVariable outputVariable <p>Extensions</p> <p>More Information</p>
<p>Receive</p> <p>Allows the business process to do a blocking wait for a particular message to arrive.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> correlations fromPart <p>Supported Attributes</p> <ul style="list-style-type: none"> partnerLink portType operation Variable createInstance messageExchange <p>Extensions</p> <p>More Information</p>

Features	Support
<p>Reply</p> <p>Returns a message from the process to the same partner that initiated the operation. The combination of Receive and Reply activities creates a request-response operation.</p>	<p>Supported Elements</p> <ul style="list-style-type: none">correlationstoPart <p>Supported Attributes</p> <ul style="list-style-type: none">partnerLinkportTypeoperationVariablefaultNamemessageExchange <p>Extensions</p> <p>More Information</p>

Features	Support
<p>PartnerLink</p> <p>Identifies the parties that interact with your business process. Each link is defined by a partner link type and a role name.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <ul style="list-style-type: none"> name partnerLinkType myRole partnerRole <p><i>Not Supported</i></p> <ul style="list-style-type: none"> initializePartnerRole <p>Extensions</p> <p>More Information</p>
Basic Activities	
<p>Assign</p> <p>Assigns values to variables. You use the Assign element to copy data from one variable to another, construct and calculate the values of expressions, and store new data in variables. Expressions are required to perform simple computation or operate message selections, properties, and literal constants to produce a new value for variables.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> copy (child elements from and to) extensibleAssign <p>Supported Attributes</p> <ul style="list-style-type: none"> from (name="variable" type="NCName") from (name="part" type="NCName") to (name="variable" type="NCName") to (name="part" type="NCName") validate from (name="expressionLanguage" type="anyURI" — xpath only) from (name="property" type="QName") from (name="partnerLink" type="NCName") from (name="endpointReference" type="bpws:tRoles") from (name="opaque" type="bpws:tBoolean") to (name="queryLanguage" type="anyURI") to (name="property" type="QName") to (name="partnerLink" type="NCName") <p>Extensions</p> <p>More Information</p>

Features	Support
<p>JavaScript</p> <p>Acts like an Assign activity that enables you to use JavaScript (E4X), rather than using XPath 1.0.</p>	<p>Supported Elements copy (child elements from and to) extensibleAssign</p> <p>Supported Attributes Validate</p> <p>Extensions</p> <p>More Information</p>
<p>Validate</p> <p>Validates the values of variables against their associated XML and WSDL data definition. The element includes a Variables property that lists the variables for the process, and allows you to specify which variables to validate. When one or more variables prove invalid against a corresponding XML definition, a standard fault, <code>bpel:invalidVariables</code>, is thrown.</p>	<p>Supported Elements</p> <p>Supported Attributes variable</p> <p>Extensions</p> <p>More Information</p>
<p>Empty</p> <p>Used as a placeholder within a process to catch and suppress faults or to help synchronize actions within a flow activity that are executed concurrently.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Wait</p> <p>Waits for a specified time or until a deadline is reached.</p>	<p>Supported Elements for until</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Throw</p> <p>Used to signal a specific internal fault, and can provide a QName and information for that fault.</p>	<p>Supported Elements</p> <p>Supported Attributes faultName faultVariable</p> <p>Extensions</p> <p>More Information</p>

Features	Support
<p>ReThrow</p> <p>Used to rethrow a fault.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Exit</p> <p>Terminates the execution of a business process instance.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Compensate</p> <p>Invokes the compensation handler of a particular scope.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>CompensateScope</p> <p>Invokes the compensation handler of a particular scope.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
Structured Activities	
<p>If</p> <p>Supports conditional behavior of a business process instance. The If activity consists of conditional branches defined by the If and Else If elements, followed by an optional Else branch. The conditions on If and Else If branches are evaluated in the order they appear. During execution, the first branch whose condition holds true is taken and provides the activity specified for the If activity. In other words, if there are several Else If branches whose conditions hold true, only the first of them will be executed.</p>	<p>Supported Elements</p> <p>elseIf condition</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>

Features	Support
<p>While</p> <p>Repeatedly execute one or more activities as long as specific conditions are in place at the beginning of each iteration.</p>	<p>Supported Elements condition</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>RepeatUntil</p> <p>Repeatedly executes one or more activities as long as specific conditions are in place after the execution of each iteration. This element contains other elements that are repeated until the success criteria you specify are met. If the condition you specify leads to true, the activities listed will be executed once.</p>	<p>Supported Elements condition</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>ForEach</p> <p>Repeatedly execute its contained scope activity exactly N+1 times where N equals the Final Counter Value minus the Start Counter Value.</p>	<p>Supported Elements startCounterValue finalCounterValue completionCondition</p> <p>Supported Attributes counterName</p> <p><i>Not Supported</i> parallel</p> <p>Extensions</p> <p>More Information</p>
<p>Pick</p> <p>Blocks a process and waits until a specified events occurs. After one of the specific event occurs, the activity associated with this event is performed. The possible events are the arrival of a message or a timer-based alarm. The selected activity is dependent upon which event occurs first.</p>	<p>Supported Elements onMessage onAlarm</p> <p>Supported Attributes variable</p> <p>Extensions</p> <p>More Information</p>
<p>Flow</p> <p>Defines a set of activities that will execute concurrently (in parallel). This is a structured activity, containing other activities separated into individual control paths or branches. You can embed as many paths in the activity as you want, and they will all be executed simultaneously.</p>	<p>Supported Elements <i>Not Supported</i> links</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>Sequence</p> <p>Use the BPEL Designer and Service Engine. Activities within a sequence execute in strict sequential order, completing when the last activity within the nest has finished.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>

Features	Support
<p>Scope</p> <p>Essentially, this activity is a collection of child activities that can have their own Variables, Fault and Event Handlers, and correlation sets. The Scope activity provides the behavior context for the child elements.</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> variables faultHandlers eventHandlers <p><i>Not Supported</i></p> <ul style="list-style-type: none"> partnerLinks correlationSets compensationHandler terminationHandler <p>Supported Attributes</p> <p><i>Not Supported</i></p> <ul style="list-style-type: none"> isolated exitOnStandardFault <p>Extensions</p> <p>More Information</p>
Other Activities	
<p>OnMessage</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <ul style="list-style-type: none"> variable <p>Extensions</p> <p>More Information</p>
<p>OnMsgCommon</p>	<p>Supported Elements</p> <ul style="list-style-type: none"> fromPart correlations <p>Supported Attributes</p> <ul style="list-style-type: none"> partnerLink portType operation messageExchange <p>Extensions</p> <p>More Information</p>
<p>OnAlarmPick</p> <p>Specifies an event that is triggered when a given duration variable is exceeded.</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>

Features	Support
<p>CompletionCondition</p>	<p>Supported Elements Supported Attributes countCompletedBranchesOnly</p> <p>Extensions</p> <p>More Information</p>
<p>Catch</p> <p>Used to intercept a specifically defined type of fault.</p>	<p>Supported Elements faultName faultVariable faultMessageType faultElement</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>
<p>OnEvent</p> <p>Indicates that a specified event is triggered when a message arrives.</p>	<p>Supported Elements messageType variable</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>

Features	Support
<p>Activity</p>	<p>Supported Elements <i>Not Supported</i> targets sources</p> <p>Supported Attributes name <i>Not Supported</i> suppressJoinFailure</p> <p>Extensions</p> <p>More Information</p>
<p>property Defines a unique name and associates it with an XML Schema simple type.</p>	<p>Supported Elements</p> <p>Supported Attributes name type element</p> <p>Extensions</p> <p>More Information</p>

Features	Support
<p>propertyAlias Defines a globally named property as an alias.</p>	<p>Supported Elements query</p> <p>Supported Attributes propertyName messageType part</p> <p><i>Not Supported</i> queryLanguage (xpath only)</p> <p>Extensions</p> <p>More Information</p>
<p>PartnerLinkType Expresses the dependences between services by defining each service's role.</p>	<p>Supported Elements role</p> <p>Supported Attributes name</p> <p>Extensions</p> <p>More Information</p>
<p>Role Specifies one WSDL portType.</p>	<p>Supported Elements</p> <p>Supported Attributes name portType</p> <p>Extensions</p> <p>More Information</p>
<p>CorrelationWithPattern</p>	<p>Supported Elements</p> <p>Supported Attributes</p> <p>Extensions</p> <p>More Information</p>

Understanding the BPEL Module Project

The BPEL Module project is a group of source files which includes BPEL files, WSDL files, and XML schema files. Within a BPEL Module project, you can author a business process compliant with the WS-BPEL 2.0 language specification.

The BPEL Module project provides point-and-click support for the following:

- Using the New Project wizard to create a BPEL Module project and a Composite Application project.

- Importing WSDL Resources to act as partner services in the business process.
- Creating new WSDL resources, as needed.
- Importing XML Schema resources.
- Adding BPEL activities to the business process diagram; further defining the elements by using Property Editor dialog boxes, Properties window, and pop-up menu actions.
- Creating and changing the source code of the BPEL, WSDL and XSD files.
- Checking and validating XML source code.
- Building and adding the project as a JBI module to a Composite Application project.
- Test running BPEL processes by sending sample messages to the deployed process or processes.
- Debugging deployed business processes

Steps to Create a BPEL Module Project

Accordingly, the typical procedure to follow when building a BPEL process is:

1. [“Creating a new BPEL Module Project” on page 38](#) using the New Project wizard.
2. [“Creating a Composite Application Project” on page 45](#). For sample processes, Composite Application projects are created automatically for you. For the processes created from scratch, you create the Composite Application project manually.
3. [Add JBI Modules](#) to the Composite Application project.
4. (Optional) Build the Composite Application project and make sure that the Application Server is started.
5. [Build and Deploy the Composite Application Project](#) the Composite Application project to the BPEL Service Engine.
6. [Create test cases](#).
For sample processes, test cases are automatically created; for new projects, you need to create at least one test case.
7. [Run](#) one or all test cases.
8. (Optional) [Debug the BPEL process](#).

Creating Sample Processes in the BPEL Designer

The best way to get acquainted with constructing BPEL diagrams is to create sample processes. You can design your BPEL process by modifying existing sample processes.

For samples, the New Project sample wizard automatically generates both types of projects, BPEL Module and Composite Application, so you do not need to separately create each of these projects. The IDE automatically adds the sample BPEL Module project as a JBI module to the Composite Application project.

In the BPEL Designer, you can create the following sample processes:

- A Synchronous Sample Process
- An Asynchronous Sample Process
- Travel Reservation Service sample

A Synchronous Sample Process

A synchronous process refers to a conversation style in which the client sends a message to the process, waits for a reply, and continues work only when the reply comes back. When you create a synchronous sample process, the IDE generates a skeletal process with a single synchronous operation and the required WSDL and XML schema files.

An Asynchronous Sample Process

An asynchronous process applies to long-running conversations in which the client does not wait for a reply from the process before continuing its work. Instead of returning the result synchronously to the client, this process accepts the client's request, performs work that might be long-running, and then asynchronously calls back to the client when the work is done. When you create an asynchronous process, the IDE generates a skeletal process with one incoming and one outgoing asynchronous operation and the required WSDL and XML schema files.

Note that any particular process can consist of an arbitrary collection of synchronous and asynchronous interactions with one or more conversational partners.

Travel Reservation Service Sample

This sample is a real-world BPEL process sample constructed using the majority of BPEL elements and several partner web services.

Together with the Travel Reservation Service sample, the wizard creates another project, Reservation Partner Services, a basic EJB and JMS based implementation of the three partner services.

Creating a Sample BPEL Module Project

The following steps describe the general flow for creating a new project from a sample BPEL module project.

▼ To create a sample BPEL Module project:

- 1 Choose File → New Project (Ctrl-Shift-N).
- 2 In the Categories list, expand the Samples node and select SOA.
- 3 In the Projects list, select the sample project you want to create and click Next.

- 4 In the Name and Location page, name the project and specify the location of project files.
- 5 Click Finish.

The wizard creates two types of projects for the selected sample: a sample BPEL Module project and a sample Composite Application project. You are free to modify the sample business process and or add additional BPEL processes to the BPEL Module. To deploy, test-run, and debug the BPEL process, use the Composite Application project.

Navigating in the BPEL Designer

This section explores the navigation capabilities of the BPEL Designer.

[“The BPEL Editor Views” on page 27](#)

[“The Navigator Window” on page 29](#)

[“The Properties Window” on page 32](#)

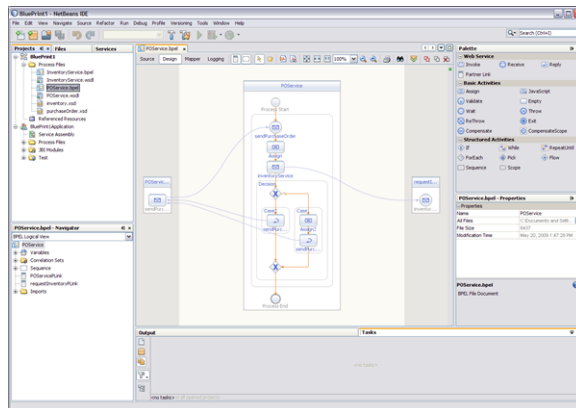
[“Scrolling” on page 32](#)

[“Zooming In and Out of the Diagram” on page 33](#)

[“Printing BPEL Diagrams and Source Files” on page 33](#)

The BPEL Designer Window

The new BPEL file opens in the Design view of the BPEL Designer.



To open the BPEL Designer, either create a new BPEL Process or open an existing BPEL Process. The image above shows the NetBeans IDE open to the BluePrint1 Project's BPEL process in the BPEL Designer. If the Pallet and Properties windows are not displayed in your current view, click **Windows** → **Reset Windows** from the NetBeans menu.

The BPEL Designer includes the following components:

- **BPEL Editor** — The center pane displays the BPEL Editor. The Design view of the BPEL Editor enables you to visually model the business process. The BPEL Designer automatically generates BPEL code that corresponds to the visual design.
- **Pallet** — The Pallet, available in the designers upper-right pane of the Design view, provides easy access to the BPEL elements.
- **Properties Window** — The Properties window, available in the lower-right pane of the Design view, provides the property sheet for any selected component or activity.
- **Navigator Window** — The Navigator window, available in the lower-left pane, shows the BPEL Logical View of the BPEL process.
- **Source View** — Click the **Source** button, and the Source view of the BPEL Editor displays the code for the current process.
- **BPEL Mapper** — To see the BPEL Mapper view, select a BPEL activity in the Design view of the editor, then click the **Mapper** button. The Mapper view of the BPEL Editor provides a framework that enables you to define and direct BPEL process data.
- **BPEL Logger** — To see the BPEL Logger view, select a BPEL activity in the Design view of the editor, then click the **Logger** button. The Logger, similar in appearance to the BPEL Mapper view, enables you to select the level of logging for the various process activities.

The BPEL Editor Views

In the BPEL Editor you can switch between Source View, Design View, Mapper View and Logging View. All the views are always kept in sync.

- **Design View** — The Design view is a business processes designer where you can author a diagram of your business process. In the Design view, you add, edit, and delete diagram elements. The diagram constructed in the Design view is automatically generated into BPEL source code compliant with the WS-BPEL 2.0 specification with the exceptions listed in the BPEL 2.0 Language Constructs section of the [BPEL Service Engine User's Guide](https://open-esb.dev.java.net/kb/preview3/ep-bpel-se.html) (<https://open-esb.dev.java.net/kb/preview3/ep-bpel-se.html>).

The Design view opens by default when you double-click a BPEL source file from a BPEL Module project in the Projects window. To switch to the corresponding place in the Source view, right-click an element in the Design view and select **Go to Source** (Alt-O).

- **Source View** — The Source view shows the underlying code for a business process diagram. The Source view is based on the IDE's XML Source view and provides access to conveniences such as code folding, XML syntax highlighting, and code completion.

You can perform source level editing as well as visual designing. The BPEL Designer will perform round-trip two-way engineering to ensure that the Design view and Source view remain synchronized with each other. The IDE will automatically re-parse the BPEL source file and rebuild the diagram every time you perform manual edits of the source file.

To switch to the corresponding place in the Design view, place a cursor at the line in the Source view, right-click and choose Go to Design (Alt-D).

- **Mapper View** — The BPEL Mapper provides a framework for processing and directing BPEL process data. The BPEL Mapper can be used to assign values or to set conditions. To switch to the the Mapper view press Ctrl-Shift-F9 or click the Mapper tab on the editor toolbar. For more information refer to the [“Using the BPEL Mapper” on page 82](#) section.
- **Logging View** — The Logging view provides you with the capability to set logging or alerting rules for the process. To switch to the the Logging view press Alt-L or click the Logging tab on the editor toolbar. For more information see the [Logging and Alerting](#) section.

Cloning Document Views

The Clone Document feature is a customization option which enables you to clone documents views. For example, if you want to see both the source and the design view of a BPEL process at the same time (or the Design and Mapper view) follow the instructions below.

Several views of one document are always kept in sync.

▼ To Clone the Document View:

- 1 Open the BPEL file
- 2 Right click the tab with the file name and choose Clone Document. Another tab with the same document will be created.
- 3 Drag and drop one of the tabs to the location you choose: left, right or to the bottom of the screen. An orange frame will show you where the window you are dragging will be placed.

Element Documentation and Report Generation

The BPEL Designer includes a feature that allows you to create comments (documentation) attached to the elements of a BPEL process. This documentation is then included with the source code of the BPEL process and can later be extracted and included in a report.

Creating Documentation for an Element

1. Select an element on the diagram or in the Navigator window.

2. Click the selected element's Documentation icon which appears next to the selected element in the diagram.
A documentation window appears.
3. Type any information or comments that you feel are useful, into the Documentation window. This documentation is now available to you whenever you open the Documentation window, and is also written to the element's Properties file.

Generation a Report

The Report Generation feature of the BPEL Designer enables you to generate a PDF document describing the BPEL Process. By creating a custom report you can create a more verbose report or include information for only those elements that have documentation, in the report.

1. From the BPEL Diagram toolbar, press the Generate Report button. The report, in PDF format, is added to the project's Process files in the Projects window.

The report includes the following information:

- The name of the process
 - The diagram
 - Information about partner links
 - Information about imported documents
 - A list of all defined variables
 - Information about the process elements
 - Documentation created for the process elements
2. To customize a report so it includes all of the element properties, or to include only elements that have documentation, click the Customize Report button in the BPEL Diagram toolbar.

A Customize Report dialog box appears.

- Choose Generate Verbose Report to include all of the element properties in the report.
 - Choose Include Only Elements with Documentation to only include elements with documentation in the report.
3. Click OK, and click the Generate Report button. If a report already exists, the new report will overwrite the existing report in the Projects window.

The Navigator Window

The Navigator window is a companion of the BPEL Designer. If the Navigator window is not visible, you can manually invoke it by selecting Window → Navigating → Navigator from the main menu or pressing the Ctrl-7 key combination.

The Navigator window provides two distinct views of the BPEL process: BPEL Logical View and XML View. You can switch between the XML View and BPEL Logical View using the drop-down menu in the upper part of the Navigator window.

XML View

The XML View is identical to the Navigator view that is available for all XML documents opened in the IDE. The XML View is a companion to the BPEL Source view. Double-click any Navigator node and the Source view adjusts the current line of code to show the selected element.

Logical View

The Navigator also provides the BPEL Logical View of BPEL processes. When you select BPEL constructs in the Design view, the BPEL Logical View shows the same element selected. Alternatively, when you select a node in the BPEL Logical View's tree, the corresponding element is selected on the diagram.

Right-clicking the nodes in the BPEL Logical View invokes pop-up menus with actions relevant to the particular node. For example, for the Assign element, the actions are Go to Source, Go to Design, Wrap With, Move Up and Move Down, Toggle Breakpoint, Delete, Show BPEL Mapper, and Properties. The Go to Source and Go to Design actions, available for most of the nodes, have associated keyboard shortcuts: Alt-O for Go to Source and Alt-D for Go to Design.

In general, the nodes in the Navigator window correspond to the elements on the diagram. In addition, there are nodes, such as Variables and Correlation Sets, that are related to functionality not directly accessible from the diagram.

To view the variables used in the business process, expand the Variables node in the BPEL Logical View of the Navigator window. For variables, the following commands are available in the pop-up menu:

- **Go To Source.** Opens the source of the BPEL file and places the cursor at the place where the variable is mentioned for the first time.
- **Go To Type Source.** Opens the source file that contains a definition of the variable type. This can be, for example, a WSDL file.
- **Find Usages.** Shows usages of variables in the BPEL file. This command is also available from the pop-up menu for correlation sets and Partner Link elements.

Of particular relevance is the Imports node, which lists XSD and WSDL files referenced with the help of the Import element in your BPEL file. Using the pop-up menu for the Imports node, you can add reference to an XSD or WSDL file. Note that only files located in the project folder may be referenced.

▼ To add a resource file (WSDL or XSD) as an import:

- 1 In the BPEL Logical View of the Navigator window, right-click the Imports node and choose one of the following, depending on the format of the imported file: Add WSDL Import or Add Schema Import.

- 2 In the **Create New Import dialog box**, select the file in your project structure to add it as import.

Note – Before you can import a file, you must first add the files stored in your project directory to the project structure, then you can add them as imports. The files that are already referenced are displayed in the strikethrough style.

- 3 View the values in the read-only **Namespace and Type fields** and click **OK**.

The resource file you have just added appears under the Imports node in the Navigator window.

▼ **To add a property to a WSDL file:**

From the Navigator window you can add properties and property aliases to the WSDL files referenced in the BPEL document.

- 1 In the **BPEL Logical View of the Navigator window**, right-click a WSDL file under the **Imports** node and choose **Add Property** from the pop-up menu.
- 2 In the **Create New Correlation Property dialog box**, specify the property name.
- 3 Select the property type and click **OK**.

▼ **To add a property alias to a WSDL file:**

- 1 In the **BPEL Logical View of the Navigator window**, right-click a WSDL file under the **Imports** node and select **Add Property Alias** from the pop-up menu.
- 2 In the **Create New Property Alias dialog box**, click **Browse** next to the **Property** field to specify the property.
- 3 In the **Property Chooser dialog box**, select the property for which you are creating the alias and click **OK**. The **Property Type** field in the **Create New Property Alias dialog box** is populated with the type.
- 4 In the **Map Property To tree**, expand the WSDL file node and select the message or message part.
- 5 **To add a query, enter the query string in the Query text field.**
If the **Synchronous with Tree** checkbox is selected, the **Query** field is updated each time you change the selection in the **Map Property To tree**.
- 6 Click **OK**.

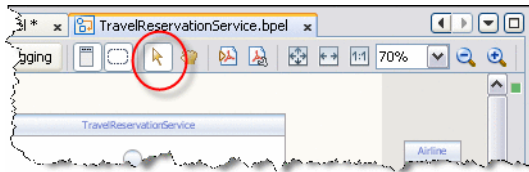
See Also For more information on defining properties and property aliases with the WSDL Editor, refer to *Using the WSDL Editor*.

The Properties Window

The Properties window contains the properties information for the currently selected element of the process. You can also use the IDE's Properties window to configure all BPEL element properties. The contents of the Properties window differs depending on the active element of the process. To open the Properties window, choose Window → Properties or press Ctrl-Shift-7.

Scrolling

When you open a BPEL file from the Projects window, the diagram opens in the Editing Mode of the Design view by default. In this mode, you can edit the diagram and scroll through it. The Editing Mode is enabled when the Navigation mode is selected on the Editor toolbar.



In the Editing Mode, you can scroll through the diagram by using the following methods:

- Turning the mouse wheel
- Using the horizontal and vertical scroll bars
- Using the thumbnail view to select the section of the diagram to display. To access the thumbnail view, click the Thumbnail button located below vertical scroll bar, or press Ctrl+B
- Pressing the Tab key to move through elements

Collapsing and Expanding Process Blocks in the Diagram

The diagram enables you to collapse and expand process blocks to allow you to focus on other processes.

▼ To collapse and expand a process block:

- 1 Click on the block's border to highlight the block. A Collapse button appears above the left corner of the block.
- 2 Click the Collapse button. The process block is displayed as a small block in the diagram flow.
- 3 To expand the process block, click the Expand icon in the center of the collapsed process block. That block is expanded.

Zooming In and Out of the Diagram

The zoom feature enables you to reduce or enlarge the size of your diagram to get a closer view or to see more of the diagram at a reduced size. You can change the zoom value using the Zoom Value drop-down list on the Editor toolbar.

- To scale the diagram to fit the window, click Fit Diagram.
- To scale the diagram width to fit the window width, click the Fit Width button.

The minimum scale size is 33% and some large diagrams might not fit entirely the window.

To change the scale do one of the following:

- Click Zoom In or Zoom Out button on the toolbar.
- Click Fit Diagram button on the toolbar to scale the diagram to fit the window.
- Click Fit Width button on the toolbar to scale the diagram width to fit the window.
- Turn on the Navigation Mode on the toolbar, then you can zoom in and out using the mouse wheel.

Printing BPEL Diagrams and Source Files

You can print BPEL diagrams and source files and customize printing settings, including border, headers, footers, colors, line numbers, and zooming, to suit your preferences.

▼ To preview and print a BPEL diagram or source file:

- 1 Open a BPEL file in the Design view.
- 2 Choose File from the main menu and select one of the following commands:
 - **Print Preview.** Preview the print layout or configure print settings.
 - **Print to HTML.** Print the .BPEL file as an HTML file.

▼ To customize print options:

- 1 In the IDE, select an object you want to print.
- 2 In the Print Preview window, click Print Options. The Print Options dialog box opens.
- 3 Change the print settings to suit your preferences:
 - **Print Border** — Adds a border to the printed page. Click the Color icon to change the border color.
 - **Print Header and Print Footer.** — Specifies the text, alignment, color, and font of the header and footer.
 - To hide the header or footer, clear the Print Header or Print Footer checkboxes, respectively.
 - To specify the header or footer pattern text, click in the field corresponding to the alignment (Left, Center, or Right) and select one of the buttons below. For example, to add the time of printing at the bottom left corner, select the Print Footer checkbox, click into the Left field, and click the "Time of printing" icon.
 - Click the Choose Footer Color and Choose Footer Font icons to modify the color and the font for the page header and footer.
 - **Line Numbers** — Specifies whether to print line numbers for source files.
 - **Wrap Lines** — Wraps the lines to fit them on the page.
 - **Print as in Editor** — The printed page will look like you see it in the editor.
 - **Text Font and Color** — Specifies the color and font of the text when you are printing, for example, source files.
 - **Background Color** — Specifies the background color.
 - **Line spacing** — Specifies the value for line spacing.
 - **Zoom** — Specifies the scale for the printed text or diagram on the page. You can select to fit width or height or choose a specific zoom scale.
- 4 Click OK.

▼ To customize page settings:

- 1 In the IDE, select an object you want to print.
- 2 Choose File → Print Preview.
- 3 In the Print Preview window, click Page Setup. The Page Setup dialog box opens.

- 4 You can also invoke the Page Setup dialog box by choosing File → Page Setup.
- 5 On the Page Setup page specify the following parameters:
 - Paper size
 - Source of the paper
 - Paper orientation
 - Margin sizes
- 6 Click Printer button and specify the printer.
- 7 Click OK.

Creating a BPEL Module Project

The proceeding sections describe the steps used to create a simple BPEL Module project. To demonstrate the procedures we are using the Synchronous sample project.

Accordingly, the typical procedure to follow when building a BPEL process is:

1. [“Creating a new BPEL Module Project” on page 38](#) using the New Project wizard. Creating the BPEL module project also includes:
 - a. Creating an XML Schema using the XSD Editor.
 - b. Creating the WSDL Documents using the New WSDL Editor.
 - c. Create the BPEL Processes using the BPEL Designer.
2. [“Creating a Composite Application Project” on page 45](#).

When new projects are created using the sample projects that are included with GlassFish, the Composite Application projects are created automatically. When you create a project from scratch, you need to create the Composite Application project manually.
3. [Add JBI Modules](#) to the Composite Application project.
4. (Optional) Build the Composite Application project and make sure that the Application Server is started.
5. [Build and Deploy the Composite Application Project](#) to the BPEL Service Engine.
6. [Create test cases](#).

For sample processes, test cases are automatically created; for new projects, you need to create at least one test case.
7. [Run](#) one or all test cases.
8. (Optional) [Debug the BPEL process](#).

Software Requirements and Installation

The following sections walk you through the Synchronous sample project to demonstrate the steps involved in creating and running a BPEL Module project.

It is assumed that you have installed the following:

- [GlassFish ESB Installation](#), which includes the following:
 - GlassFish v2 Update Release 2 (UR2)
 - NetBeans IDE 6.1 ML
 - Open ESB core components
 - Java Business Integration (JBI) service engines
 - Java Business Integration (JBI) binding components
 - Java Business Integration (JBI) component tooling
- [JDK \(Java Development Kit\) 6](#)

Note – You must have the JDK (Java Development Kit) software installed and JAVA_HOME set prior to installing GlassFish ESB or the Installer will halt the installation. See [Installing the JDK Software and Setting JAVA_HOME](#) for details.

To get the latest installation of GlassFish ESB, see [GlassFish ESB Downloads](#).

To download Shared Libraries, JAR files, or NetBeans-module files for BPEL SE, see [BPEL Service Engine Downloads](#).

Starting GlassFish

The BPEL Service Engine starts together with GlassFish. Before deploying and performing test runs of a Composite Application project in the NetBeans IDE, make sure that the GlassFish Application Server is started.

▼ To check the Status of the GlassFish V2 Application Server in the NetBeans IDE

1 If the Services window is not visible, choose `Window` → `Services`.

2 In the Services window, expand the `Servers` node.

The Servers node should contain a `GlassFish V2` subnode. If the `GlassFish V2` node does not appear, go to “[To Register the GlassFish V2 Application Server with the NetBeans IDE](#)” on [page 37](#):

If a green arrow badge appears on the GlassFish V2 node, the server is running. If a green arrow badge does not appear, go to [“To Start the GlassFish V2 Application Server in the NetBeans IDE” on page 37.](#)

▼ To Register the GlassFish V2 Application Server with the NetBeans IDE

- 1 If the Services window is not visible, choose Window → Services.**
- 2 In the Services window, right-click the Servers node and choose Add Server from the pop-up menu.**

The Add Server Instance dialog box opens.
- 3 In the Choose Server page of the dialog box, select GlassFish V2 from the Server field.**

The Add Server Instance dialog box opens.
- 4 (Optional) In the Name field, change the default name for the server.**

The IDE uses this name to identify the server.
- 5 Click Next.**

The Platform Location Folder page opens.
- 6 In the Platform Location field, click Browse and select the installation location of the application server.**
- 7 Select the Register Local Default Domain radio button and click Next.**
- 8 Enter the user name and password for the domain's administrator.**

If you accepted the default values during the installation, the user name is admin and the password is adminadmin.
- 9 Click Finish.**

▼ To Start the GlassFish V2 Application Server in the NetBeans IDE

- 1 In the Services window, right-click the GlassFish V2 node and choose Start.**
- 2 Wait for the following message to appear in the Output window:**

Application server startup complete.

When the server is running, the IDE displays a green arrow badge on the GlassFish V2 node.

The BPEL Service Engine is represented as sun-bpel-engine in the Services window of the IDE, under the GlassFish V2 → JBI nodes.

Creating a new BPEL Module Project

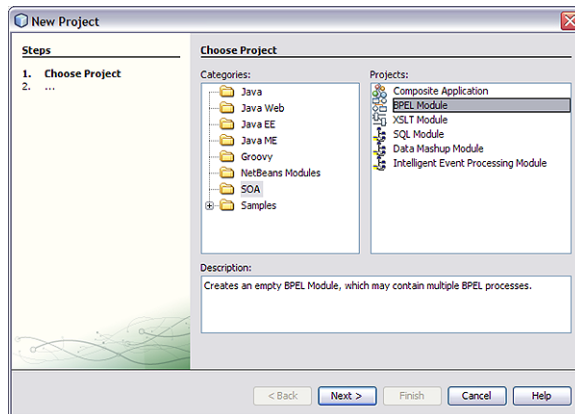
The following sections use the Synchronous sample project, and provide step-by-step directions for creating a simple BPEL module project, using the Synchronous sample project.

▼ Create a BPEL Module project

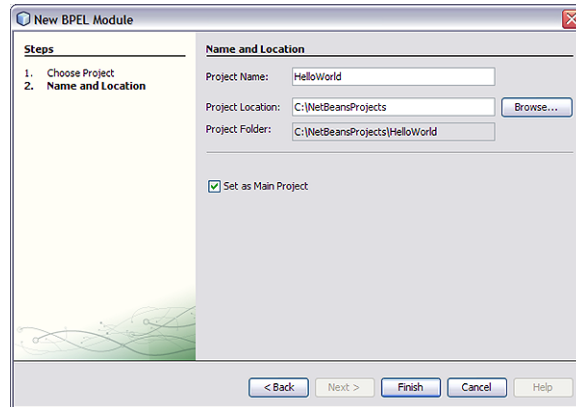
- 1 In the NetBeans IDE, choose **File > New Project**.

The New Projects wizard appears.

- 2 Under **Categories**, select **Service Oriented Architecture**.
- 3 Under **Projects**, select **BPEL Module** and click **Next**.

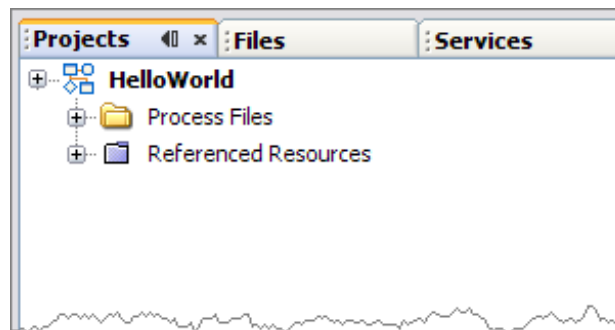


- 4 In the **Name and Location** page, enter the project name (for this example **HelloWorld**) and specify the project location or accept the defaults.



5 Click Finish.

The Projects window now contains a project node for the BPEL Module project.



Creating the XML Schema and the WSDL Document

Generally, the steps used to create a BPEL module project are:

1. Create a New BPEL Project
2. Create the XML Schema or XSD file
3. Create the WSDL Documents
4. Create the BPEL Process

The XSD file (XML Schema) helps to define the projects message structure or operations. Complex message structures are defined in the XSD file and imported into the WSDL Document. The WSDL Documents define the interfaces for the project. The BPEL Designer enables you to graphically design your BPEL business process.

To create the XML Schema and WSDL Document for this tutorial, following the directions provided in [Creating a “Hello World” Composite Application](#). After you have created the XML Schema and WSDL, return here.

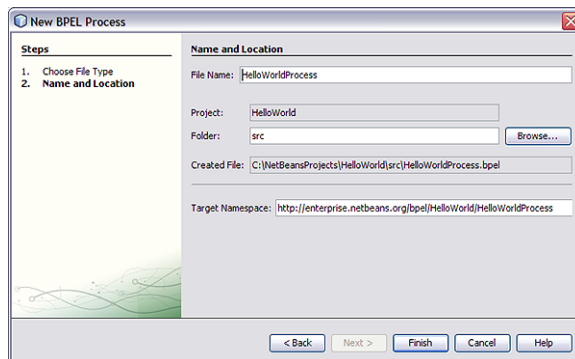
Creating a BPEL Process Using the BPEL Designer

Now that you have created your XML Schema and WSDL Document, you can create your BPEL process.

▼ Create the BPEL Process

- 1 In the Projects window, expand your BPEL module project node, right-click the Process Files node, and choose **New** → **BPEL Process** from the pop-up menu.

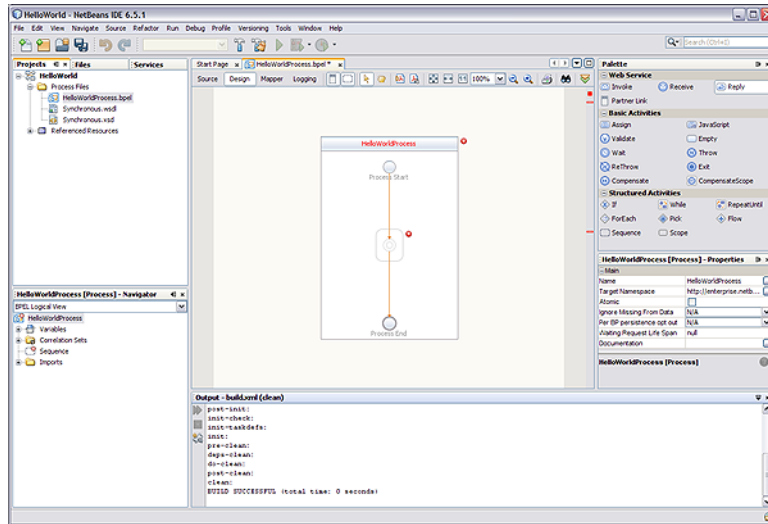
The New BPEL Process dialog box appears.



- 2 Enter a name for the process file name (HelloWorldProcess for this example), and click **Finish**.

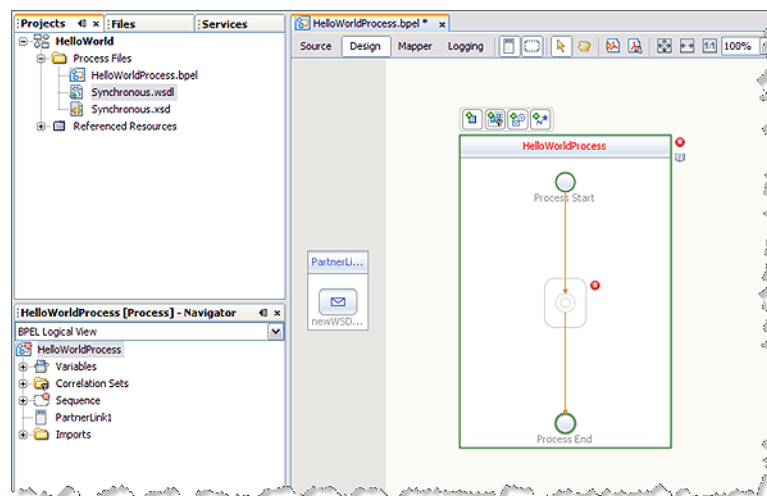
The new BPEL file opens in the Design view of the BPEL Designer.

If the Palette and Properties windows are not displayed in your current view, click **Windows** → **Reset Windows** on the NetBeans menu.



3 Add a partner link to the BPEL Process.

- a. In the Projects window, expand your project's Process Files node and select the .wsdl file (Synchronous.wsdl for this example).
- b. Drag the .wsdl file from the Projects window to the left side of the Design view canvas. The IDE provides visual prompts to show you where you can drop the selection. The BPEL Editor adds a partner link to the canvas.



4 Add a Receive activity to the BPEL Process.

a. From the Web Service section of the Palette window, select the Receive activity.

b. Drag the Receive activity to the HelloWorldProcess process box in the Design view canvas, between the Process Start and the Process End activities.

The Receive1 activity is added to the process box.

c. Click the Receive1 activity's Edit icon.

The Receive1 Property Editor appears.

d. On the Main tab, change the value in the Name field to start.

e. From the Partner Link drop-down list, choose PartnerLink1.

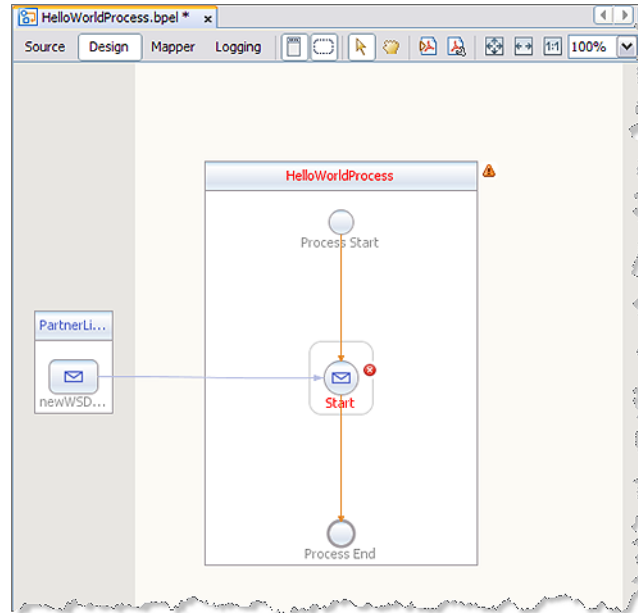
The IDE populates the Operation field with NewWSDLOperation.

f. Click the Create button next to the Input Variable Field.

The New Input Variable dialog box appears. Click OK to accept the default values.

g. Click OK to close the Receive 1 Property Editor.

The Design view displays the new connection between PartnerLink1 and the Start activity in the process box.



5 Add a Reply activity to the BPEL Process.

- a. **Select the Reply activity in the Web Service section of the Palette. Drag and drop the Reply to the prompt between the Start activity and the Process End activity in the process box on the design view canvas.**

A Reply1 activity is added to the design view canvas.

- b. **Click the Reply1 activity's Edit icon.**

The Reply1 Property Editor appears.

- c. **On the Main tab, change the value in the Name field to End.**

- d. **From the Partner Link drop-down list, choose PartnerLink1.**

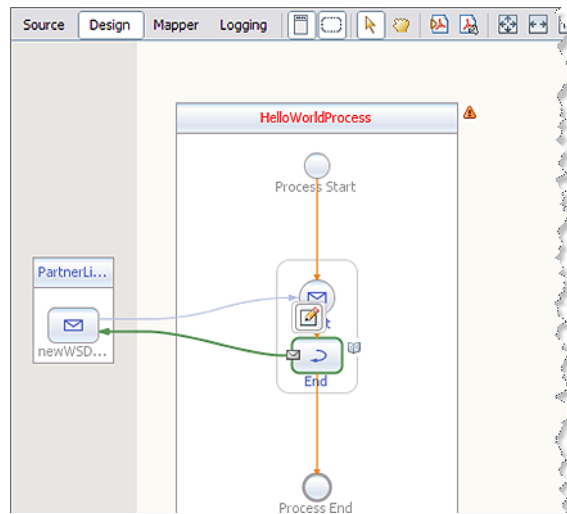
The IDE populates the Operation field with NewWSDLOperation.

- e. **To create a new output variable, make sure that Normal Response is selected, and click the Create button next to the Input Variable Field.**

The New Input Variable dialog box appears. Click OK to accept the default values.

- f. **Click OK to close the Reply1 Property Editor.**

The Design view displays the new connection between the End activity in the process box and PartnerLink1.



6 Add a Assign activity to the BPEL Process.

- a. **Select the Assign activity in the Basic Activities section of the Palette. Drag and drop the Assign to the prompt between the Start activity and the End activity in the process box on the design view canvas.**

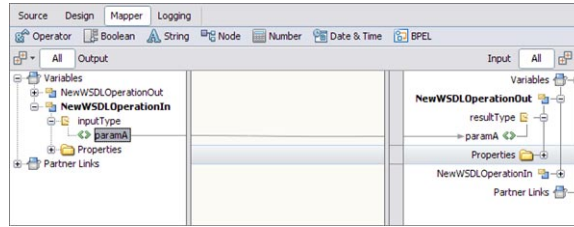
The Assign1 activity is added to the design view canvas.

- b. **Select the Assign1 activity and click the Mapper button on the editors toolbar.**

The BPEL Mapper appears.

- c. **Map the paramA node under Variables → NewWSDLOperationIn → inputType in the Output pane of the BPEL Mapper, to the paramA node under Variables → NewWSDLOperationOut → resultType in the Input pane of the Mapper. To do this, select the paramA node under Variables → NewWSDLOperationIn → inputType in the Output pane, and drag your cursor to the paramA node under Variables → NewWSDLOperationOut → resultType in the Input pane.**

This assignment copies the input statement into the output.



- 7 To save your changes click the Save All icon in the IDE menu bar.

Creating a Composite Application Project

A BPEL Module project is not directly deployable. You must first add a BPEL Module project, as a JBI module, to a Composite Application project. You can then deploy the Composite Application project. Deploying the project makes the service assembly available to the application server and enables its service units to run.

▼ Create a new Composite Application project

- 1 Choose File > New Project (Ctrl-Shift-N).
- 2 In the Categories list choose Service Oriented Architecture, in the Projects list choose Composite Application, and click Next.
- 3 In the Name and Location page, change the project name to HelloWorldApplication, and specify the location of project files.
- 4 To set the new Composite Application the main project as main, leave the Set as Main Project checkbox selected, and click Finish.
- 5 To add the BPEL Module as a JBI module to the Composite Application project, right-click the new Composite Application and choose Add JBI Module from the pop-up menu.

The Select Project dialog box opens.

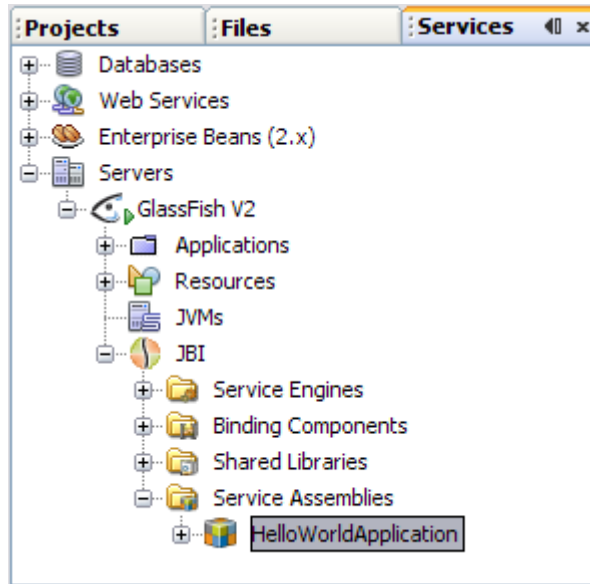
- 6 Select the HelloWorld project you created earlier and click Add Project JAR Files. The Select Project dialog box closes and the HelloWorld.jar file is added to the JBI Modules node of the HelloWorldApplication Composite Application

Building and Deploying the Composite Application Project

Building a project compiles the BPEL source file and packages the BPEL file and web service artifacts, including WSDL and XSD files, into a JAR archive. Deploying the project compiles the files in the Composite Application project, packages the compiled BPEL and related web service artifacts (including WSDL and XSD files) into an archive, and deploys them to the Application Server.

▼ Build and deploy the Composite Application Project:

- 1 Right-click the Composite Application project's node, and choose Build from the pop-up menu.**
When the build is complete the Output window reports Build Successful. If the Output window is not visible, choose Window → Output → Output.
- 2 Right-click the Composite Application project's node, and choose Deploy.**
- 3 Deployment has succeeded when you see a Build successful message in the GlassFish tab of the Output window.**
- 4 Open the Services window and expand Servers → GlassFish V2 → JBI → Service Assemblies to see your new deployed Service Assembly.**
If you do not see the deployed project, right-click the Service Assemblies node and choose Refresh.



Testing the Composite Application

You can test your Composite Application project by adding test cases, binding to the operation, supplying input, and then using the tester.

▼ Test the HelloWorldApplication Composite Application project

- 1 In the Projects window, expand the HelloWorldApplication project node, right-click the Test node, and choose New Test Case from the pop-up menu.

The New Test Case wizard opens.

- 2 Accept the default test case name, TestCase1, and click Next.
- 3 From the Select the WSDL Document page, expand the HelloWorld - Proecss Files node, select Synchronous.wsdl, and click Next.
- 4 From the Select the Operation to Test page, select the Operation1 and click Finish.

A new TestCase1 node is added under the project's Test node in the Projects window, containing two subnodes, Input and Output.

The Source Editor appears containing the Input file, Input.xml

Note – If the Source Editor does not contain a tab for Input.xml, double-click the Input node in the Projects window to open the file.

5 From the Input.xml tab of the Source Editor, do the following:

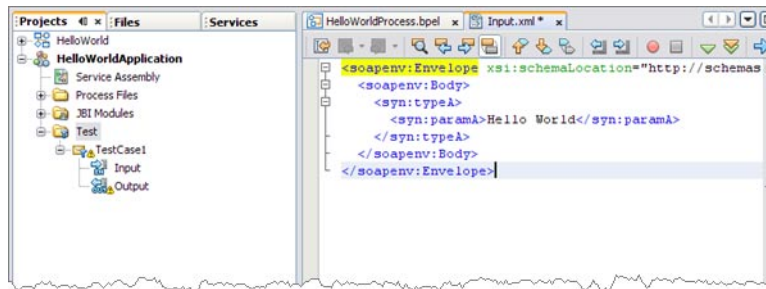
a. Locate the line:

```
<syn:paramA?>string?</syn:paramA>
```

b. Replace ?string? with Hello World, so that the line appears as follows:

```
<syn:paramA>Hello World</syn:paramA>
```

c. From the NetBeans IDE menu bar, click the Save All button.

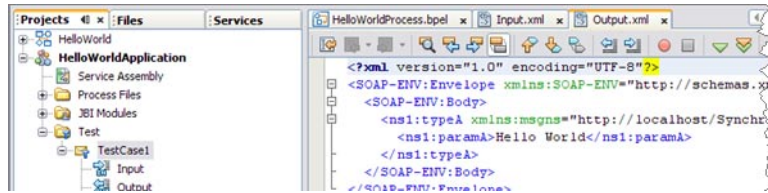


6 In the Projects window, double-click the Output node under Test → TestCase1.

Output.xml is opened in the Source Editor. Initially, Output.xml is empty until the first test run populates the file.

7 In the Projects window, right-click the TestCase1 node and choose Run from the pop-up menu.

When the Overwrite Empty Output dialog box appears, click Yes to accept new output. The first test run populates the Output.xml file displayed in the Source Editor.



The test compares the output to the contents of the output file. Because the first run of the test has nothing to compare itself to, the first test fails. Subsequent test runs will compare their output with the contents of Output.xml and should succeed.

8 Run the test again.

The testcase is compared to the current output file and succeeds.

Summary

You just created the HelloWorld, synchronous sample project. This sample project is available in the NetBeans IDE's samples file under Project/Samples/SOA/Synchronous BPEL Process.

This tutorial above demonstrates how to:

- Create a Bpel Module project
- Use the BPEL Designer to create a BPEL process
- Build and deploy a Composite Application project to GlassFish
- Create and run test cases

Developing a BPEL Process Using the Diagram

This section includes the following topics for getting started with the diagram:

- [“The BPEL Diagram” on page 50](#)
- [“Configuring Element Properties in the Design View” on page 51](#)
- [“Finding Usages of BPEL Components” on page 51](#)
- [“Saving Your Changes” on page 52](#)

The BPEL Diagram


The BPEL diagram (BPEL Design View) is the visual representation of the BPEL Process. On the diagram you can author business process by adding and configuring activities. You can also edit existing .bpe1 files. To open a .bpe1 file double-click its name in the Projects window. By default, the process diagram will be open.

In the Design view, you can perform the following operations on elements:

- **Create elements by dragging elements from the Palette to the diagram.** The Design view supports the notion of "drop points", which means that you must align elements with these drop points when you drag them. Not all elements are created via drag-and-drop from the Palette. These other elements are created using pop-up menus which are invoked when you right-click an existing diagram element.
- **Select elements on a diagram.** A single click on an element selects it. Selection is a necessary step in performing several other operations, such as deleting, moving, or editing an element.
- **Invoke pop-up menu actions for diagram elements.** Each BPEL element has a pop-up menu. This pop-up menu can be invoked by right-clicking the element. The pop-up menu will offer a set of actions which are relevant to the selected element.
- **Move diagram elements.** You can move diagram elements by selecting the element and then dragging it to a new location. If you move a container element, all its children move along with the container.
- **Edit element names in the Design view directly.** Double-click the element name on the diagram to edit it.
- **Invoke XML Validation.** You can invoke XML validation by clicking the Validate XML button on the Design view editor toolbar. For more information, see the Validation section.
- **Apply filters to diagram elements.** The Editor toolbar contains the Show Partner Links and Show Sequences toggle buttons. Both Partner Link elements and Sequence elements are shown by default. Clicking the Show Partner Links button hides the Partner Link elements on the diagram. Clicking the Show Sequences button hides the Sequence containers on the diagram. Clicking either button a second time will again reveal the Partner Link elements, or the Sequence elements on the diagram, respectively.

Note – You cannot add new Partner Link or Sequence elements to the diagram if you chose not to show them.

- **Finding elements on the diagram.** You can find BPEL elements in the Design view by their names or types. You can use either the Find bar (Edit > Find or Ctrl-F) or the Advanced Search feature (Edit > Advanced Search or Alt-Shift-F). In the Find bar, select the type of search you want to perform, type in the search query and click Find. In the Advanced Search dialog box, you can refine your search query and search BPEL elements by their name and/or type.

- **Collapsing or expanding elements on the diagram.** When a large diagram is open in the Design view, you can collapse or expand container elements, such as Sequence or Scope, using the quick action buttons that appear near the selected elements. By default, when you open a diagram in the Design view, you can see all container elements expanded. To expand all elements on the diagram, click the Expand All icon  on the Editor toolbar. You can use the following combinations of keys: Enter to expand the selected element, Shift-Enter to collapse the selected element, and Alt-Enter to expand all elements on the diagram.
- **Wrapping activities with container elements.** You can wrap elements with container activities in a single click. The wrap feature is useful, for example when you want to quickly place an activity inside another activity. In the Design view, right-click the activity you want to wrap, point to the Wrap With option, and select the wrapper BPEL activity.

Configuring Element Properties in the Design View

After you add BPEL activities to a diagram, you need to configure them. You can do this using either the Property Editor dialog boxes or the IDE's Properties window. Note that Property Editor dialog boxes are available only for some elements.

To open the Property Editor for an element, do one of the following:

- Right-click the element and choose Edit.
- Double-click the element.

To open the Properties window for an element, right-click the element and choose Properties. The properties for this element are displayed in the standard IDE's Properties window. If the IDE's Properties window is not open, choose Window > Properties from the main menu (Ctrl-Shift-7).

Finding Usages of BPEL Components

For BPEL files, the Find Usages command determines where the following elements are used in the associated .bpe1 files:

- Variable
- Partner Link
- Correlation set

▼ To find usages of a BPEL component:

- 1 **In the IDE, open the BPEL file (.bpe1) you want to work with.**
By default, the IDE opens the Design view for the BPEL file.
- 2 **In the Design view, select the element for which you want to see usages.**
You can also select the element in the BPEL Logical View of the Navigator window.

- 3 Right-click the element either in the Design view or in the Navigator window and choose Find Usages (Alt-F7) from the pop-up menu.**

The IDE opens the XML Usages window in the lower part of the IDE. The first time you invoke the Find Usages function, the window has no tabs. For each consequent query, the IDE adds a Find XML Usages tab that shows the usages of the component you selected.

- 4 (Optional) To go to the source for an element and double-click that element in the tree. The right part of the XML Usages windows is a visual representation of element usages throughout the project.**

Saving Your Changes

The BPEL Designer synchronizes the Design and Source views as follows:

- Changes you make in a diagram are reflected immediately in the corresponding source code.
- Changes you make in the source code are reflected on the diagram when you switch to the Design view.

To save changes in the Design or Source view, choose File → Save or press Ctrl-S.

The BPEL Designer Palette Elements

This section is the palette elements reference.

[“The Process Element” on page 54](#)

[“Using the Invoke Element” on page 55](#)

[“Using the Receive Element” on page 57](#)

[“Using the Reply Element” on page 59](#)

[“Using the Partner Link Element” on page 61](#)

[“Using the Assign Element” on page 66](#)

[“Using the JavaScript Element” on page 67](#)

[“Using the Validate Element” on page 69](#)

[“Using the Empty Element” on page 70](#)

[“Using the Wait Element” on page 70](#)

[“Using the Throw Element” on page 71](#)

- “Using the Rethrow Element” on page 72
- “Using the Exit Element” on page 72
- “Using the Compensate Element” on page 72
- “Using the CompensateScope Element” on page 73
- “Using the If Element” on page 74
- “Using the While Element” on page 75
- “Using the Repeat Until Element” on page 76
- “Using the For Each Element” on page 76
- “Using the Pick Element” on page 77
- “Using the Flow Element” on page 78
- “Using the Sequence Element” on page 79
- “Using the Scope Element” on page 80

A process diagram represents the connected elements in a business process. To edit a diagram, icons in the Design view Palette are dragged and dropped onto the BPEL diagram. These icons represent Web Service activities, basic activities, and structured activities in the business process.

Placeholders

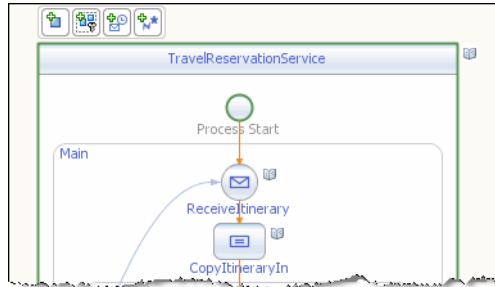
The BPEL Designer diagram provides placeholder the mark places on the diagram where you can insert an element. The behavior of element placeholders illustrates how the BPEL Designer enforces the rules of the WS-BPEL 2.0 specification.

When you drag an element from the Palette, you can see placeholders showing acceptable drop points for this element. These drop points reflect the construction logic of the diagram. As you move the mouse pointer on the diagram, a placeholder that is active for the current mouse pointer position is highlighted. Align the dragged element with one of the placeholders and release the mouse button to insert the element.

Some placeholders are always present on the diagram marking the places where it is necessary to insert an activity so that the BPEL process is valid. These are the places inside container elements, for example the If element.

The Process Element

The Process element is already present in your diagram. The New Project wizards always create a skeletal BPEL file that contains at least a process element. Therefore, the Process element is not part of the Palette. The Process element is assumed to be present, as it is the minimum requirement for a BPEL file.



Usage

1. Right-click the Process element and choose Add from the pop-up menu to add the following:
 - Variable
 - Correlation Set
 - Event Handlers
 - Fault Handlers
 - WSDL Import
 - Schema Import
2. Specify the name and the target namespace of the Process element in the Properties window, invoked by right-clicking the element and choosing Properties.

Processes

A BPEL process can be synchronous or asynchronous. A synchronous BPEL process blocks the client (the one which is using the process) until the process finishes and returns a result to the client. An asynchronous process does not block the client. Rather it uses a callback to return the result (if any). Usually we use asynchronous processes for longer-lasting processes and synchronous for processes that return a result in a relatively short time. If a BPEL process uses asynchronous web services, the process itself is usually also asynchronous.

The Web Service Elements

The Web Service section of the BPEL Designer Palette contains icons for the following business process elements:

- “Using the Invoke Element” on page 55
- “Using the Receive Element” on page 57
- “Using the Reply Element” on page 59
- “Using the Partner Link Element” on page 61

These elements represent requests, responses, and agreements between a process and its partner services.

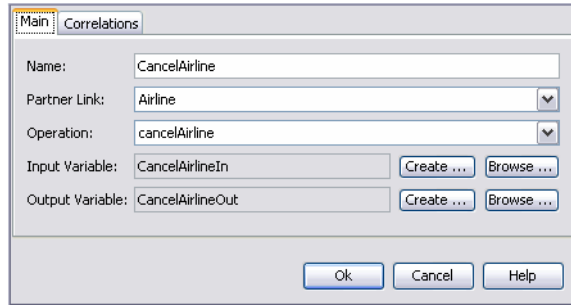
Using the Invoke Element

The Invoke element enables the business process to invoke a one-way or request-response operation on a portType offered by a partner. It enables the business process to send messages to partners. The operation is defined in the partner's WSDL file.

Usage

1. In the Design view, drag the Invoke element from the Palette to the diagram.
2. Perform one of the following procedures to associate the Invoke element with a Partner Link element:
 - Directly draw a message flow from the Invoke element to the target Partner Link.
 - Double-click the Invoke element. A dialog opens where you can examine or change the following:
 - The name of the Invoke element.
 - The partner link that is invoked.
 - The operation associated with the Invoke element.
 - The input variable associated with the Invoke element.
 - The output variable.

Both input and output variables can be created or browsed through this dialog.

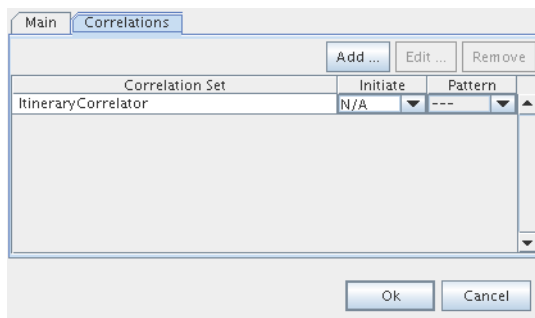


In the Property Editor dialog box, you can either create a variable or use an existing variable to hold input and output data. Click the Create button to create a variable for the Invoke element, and click Browse to choose an existing variable.

Note that when you click the Browse button, the Input Variable Chooser or the Output Variable Chooser dialog boxes opens. In these dialog boxes, a checkbox with the option to show variables with appropriate types appears. This checkbox restricts the list of available variables to those which are of the proper type for the activity you are configuring. In this way the Design view helps you develop valid BPEL code.

Correlations

Correlation sets on invoke activities, which deal with outbound operations, are used to validate that outgoing messages contain data which is consistent with the data contained within specified correlation set instances. The Correlations tab in the Invoke Property Editor dialog box enables you to examine or specify a correlation set.



The tab shows:

- Names of correlation sets
- The initiation of a correlator
- The pattern associated with the correlation

For more information see [“Understanding Correlation. Using the Correlation Wizard”](#) on page 109.

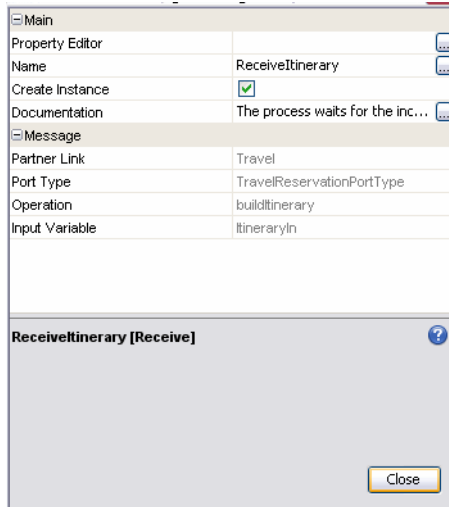
Using the Receive Element

The Receive element allows the business process to do a blocking wait for a particular message to arrive.

Usage

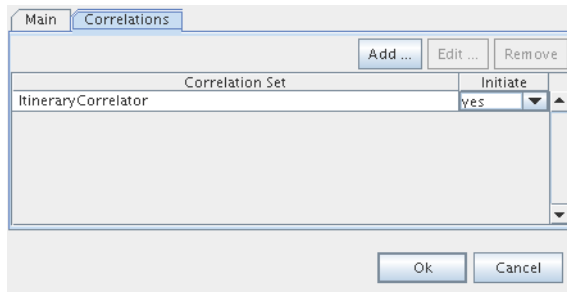
1. In the Design view, drag the Receive element from the Palette to the diagram.
2. Double-click the Receive element (or right-click it and choose Edit) to configure its properties. Here the example is provided for the Travel Reservation Service sample:
 - The name of the Receive element (ReceiveItinerary).
 - The partner link (Travel).
 - The operation associated with the Receive element (buildItinerary).
 - The input variable to the Receive element (ItineraryIn).

Select Browse for the Input Variable to open the Input Variable chooser, where you can choose other variables associated with this process. Select Create to create a new variable.
 - Create Instance. If selected, an instance of the BPEL process starts when an associated message arrives. Note that if the Receive activity is the first activity in your business process, the Create Instance checkbox must be selected.
3. You can also edit some of the element's properties in the Properties window. To open the window, right-click the Receive element and choose Properties or choose Window > Properties (Ctrl-Shift-7) from the main menu. You can edit the information by clicking on the ellipsis button. You can't edit the shadowed information from this window, to change it open the property editor as described above.



Correlations

The Correlations tab in the Receive Property Editor dialog box enables you to examine or specify a correlation set.



The tab shows:

- Names of correlation sets
- The initiation of a correlator

For more information see [“Understanding Correlation. Using the Correlation Wizard”](#) on page 109.

Using the Reply Element

Use this activity to return a message from the process to the same partner that initiated the operation. The combination of Receive and Reply activities creates a request-response operation.

This activity is used in a synchronous (request/response) operation, and specifies the same partner, port type and operation as the Receive activity that invoked the process.

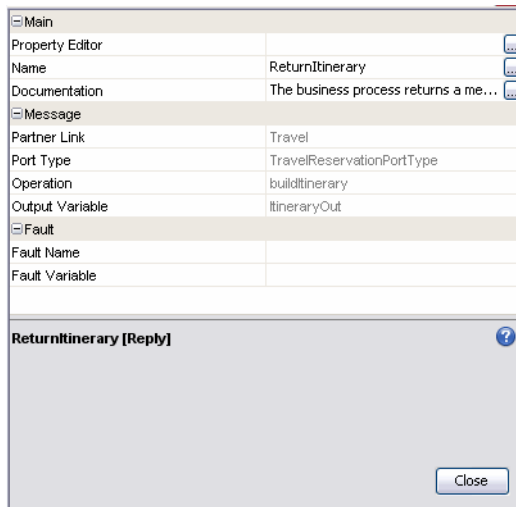
Usage

1. In the Design view, drag the Reply element from the Palette to the diagram.
2. Double-click the Reply element (or right-click and choose Edit) to open a Property Editor dialog box for the Reply element. In this dialog box, you can specify the following:
 - The name of the element.
 - The Partner Link.
 - The operation.
 - Type of response: Normal Response or Fault Response.
 - Select Normal Response if the Reply element is to be used for the normal response message type. Optionally, specify the output variable: either create a new output variable or browse for an existing variable.
 - Select Fault Response if the Reply element is to be used to send a fault message. Choose a fault name and, optionally, specify the fault variable: either create a new fault variable or browse for an existing variable.

The screenshot shows a dialog box titled "Main Correlations" with the following fields and options:

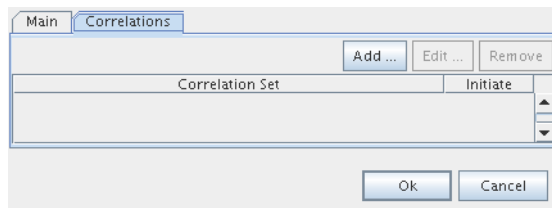
- Name:** ReturnItinerary
- Partner Link:** Travel
- Operation:** buildItinerary
- Response Type:**
 - Normal Response
 - Output Variable:** ItineraryOut
 - Buttons: Create ..., Browse ...
 - Fault Response
 - Fault Name:** tres:itineraryProblem
 - Buttons: Choose ..., Create ..., Browse ...
- Buttons:** Ok, Cancel, Help

- You can also edit some of the element's properties in the Properties window. To open the window, right-click the Receive element and choose Properties or choose Window > Properties (Ctrl-Shift-7) from the main menu. You can edit the information by clicking on the ellipsis button. You can't edit the shadowed information from this window, to change it open the property editor as described above.



Correlations

The Correlations tab in the Reply Property Editor dialog box enables you to examine or specify a correlation set.



The tab shows:

- Names of correlation sets
- The initiation of a correlator

For more information see [“Understanding Correlation. Using the Correlation Wizard”](#) on page 109.

Using the Partner Link Element

Partner Link elements identify the parties that interact with your business process. Each link is defined by a partner link type and a role name.

Partner Link Types and Roles

The type determines the relationship between a process and its partners by defining the roles played by each service in a conversation. The relationship is further determined by specifying the port type provided by each service to receive messages. Each role specifies one port type in the WSDL file.

Roles determine the conversational aspect of this process or its partner. You use a single role for a synchronous operation as the results are returned using the same operation. You use two roles in an asynchronous operation as the partner role switches during a callback.

It is easy to confuse partner links and partner link types, however:

- Partner link types and roles are special WSDL extensions defined by the BPEL specification. As such, they are defined in WSDL files, not in the process BPEL file.
- Partner Link is a BPEL 2.0 element. It is defined in the process BPEL file.

Partner link types are prerequisites to the Partner Link element definition. A Partner Link element can only be defined by referring to a particular partner link type and role which, as mentioned, must be defined in a WSDL file.

Usage

To add the Partner Link element to the BPEL process, do one of the following:

- Drag the Partner Link element from the Palette to the diagram.
- Drag a WSDL file node from the same project in the Projects window to the diagram.
- Drag a WSDL file node from a different project in the Projects window to the diagram.
- Drag a web service node from an EJB project or a Web Application project in the Projects window to the diagram.

Note: When you drag the web service node, the BPEL Designer retrieves the WSDL file from the Application Server. To successfully retrieve the WSDL file, the Application Server has to be running and the web service project must be deployed.

When you drag the Partner Link element, a WSDL file node, or a web service node to the diagram, the Partner Link Property Editor appears.

The Partner Link Property Editor

The Partner Link Property Editor dialog box enables you to establish partner links for your BPEL processes.

The Partner Link Property Editor dialog box enables you to establish partner links for your BPEL processes. The Partner Link Property Editor is invoked by double-clicking a Partner Link element on the diagram, or right-clicking the Partner Link element and choosing Edit. The Partner Link Property Editor also appears when you drag the Partner Link element, a WSDL file node, or a web service node to the diagram.

With the Partner Link Property Editor, you can specify:

- The Partner Link name
- The WSDL file associated with the Partner Link

Further on you can choose whether to use the existing partner link type or create a new partner link type.

If the WSDL file you selected contains partner link types, the Use Existing Partner Link Type option is selected and the Partner Link Type drop-down list is populated with the partner link types found in the WSDL file. You can use one of the existing partner link types or select the Use a Newly Created Partner Link Type option to create a new partner link type.

If the WSDL file does not contain partner link types, the Use a Newly Created Partner Link Type option is selected.

- **Use Existing Partner Link Type**

1. Choose the partner link type from the drop-down list. The My Role and/or Partner Role fields are filled in automatically.
2. To swap the roles of the business process itself (My Role) and the partner (Partner Role), click the Swap Roles button.

- **Use a Newly Created Partner Link Type**

1. Specify the WSDL file in which to add a partner link type. You can do one of the following:
 - Add the partner link type to the wrapper WSDL file, as suggested by the IDE in the Create in File field by default. If you choose this option, the IDE will automatically create the wrapper WSDL file in your project structure. You can use wrapper WSDL files when the original WSDL file is read-only or when you do not want to modify the original WSDL file. The original WSDL file will be imported to the newly created wrapper WSDL file.
 - Add the partner link type to a WSDL file within your project. Click Browse and locate the WSDL file in which to add the partner link type.
2. Specify the partner link type name.
3. Specify the role of the business process itself (My Role) and/or the role of the partner (Partner Role) as follows:
 - Select the appropriate checkbox.
 - Specify the role name.

- Choose the port type from the drop-down list.

You can also review and modify the Partner Link's properties in the Properties window invoked by right-clicking the element and choosing Properties.

Partner Link Layout

The partner links are placed in the left and right margins of the process diagram. Service requestors are placed on the left side, service providers are placed on the right side. To define the role and to choose the appropriate side for each partner link the IDE uses the order of roles defined for the partnerLinkType in the WSDL file. The role defined first in the partnerLinkType in the WSDL file is considered to be a service role, the second defined role is considered to be the role for the requestor and callback receiver. If the roles are defined in the reverse order in the WSDL file (the callback receiver role is defined in the first place, and the service role in the second place) then you get the improper partner link layout in the BPEL process diagram, though the operation is not damaged. If a partner link appears on the wrong side you should go to the WSDL file and swap places for the role definitions in the partnerLinkType.

Dynamic Partner Links and Dynamic Addressing

During the design-time of an application, you may need to configure certain services whose endpoints (addresses) are not known beforehand, or it may be necessary to change an endpoint

reference while the application is running. The Dynamic Partner link feature allows you to dynamically assign an endpoint reference to the partner link. This means that you can use one partner link for subsequent calls to different web-services (provided that the services use the same interface).

See [“Using Dynamic Partner Links and Dynamic Addressing”](#) on page 157 for additional use cases and more information.

Assigning an Endpoint Reference

Each partner link defines abstract information and concrete information. While abstract information, describing the web-service interface, should be static, the concrete information, such as the address and port, can be discovered and used dynamically.

Note – For successful deployment of the process, a partner link should be completely defined. When you deploy the project, the WSDL file for the partner link should contain and define both the abstract and the concrete information for the partner link, including address and port, though later the concrete information can be changed independently from the WSDL file.

Note – The BPEL specification mandates that only the partner endpoint reference (EPR) can be changed dynamically. In BPEL terms, only the partnerRole of a partner link element can have a new value assigned. The myRole value doesn't change after the BPEL has been deployed.

To assign a new endpoint reference to a partner link you can use the standard Assign activity and the BPEL Mapper.

The EPR information can be provided in several different ways:

- **Use literal to construct endpoint:** Provide the endpoint address as literal syntax and map it to the partner link.
- **Use an existing partner link's endpoint:** Extract a partner link's endpoint and assign it to another partner link in the BPEL Mapper.
- **Use an incoming message to extract the endpoint address:** Define the message with the end point schema as part of it, and use the message variable to assign the endpoint to the partner link at runtime.
- **Using a database query to provide an endpoint address** The BPEL service does a dynamic addressing invocation, with the address coming from a database.

Each of these options is explained in more detail in the advanced section of this book. For more information on each of these alternatives, see [“Using Dynamic Partner Links and Dynamic Addressing”](#) on page 157.

If you use an incoming message, an EPR schema should be defined as a part of the message in WSDL. To assign the EPR to a partner link, use the message variable.

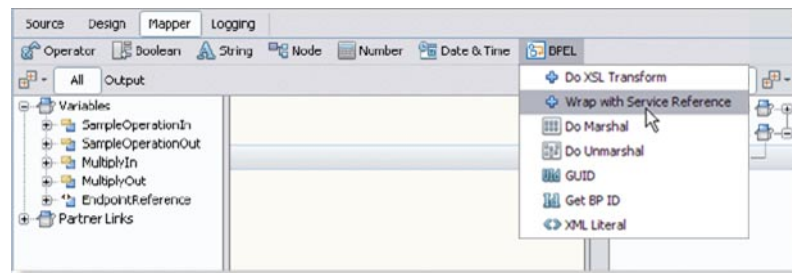
▼ To assign a new endpoint reference to a partner link from a variable:

- 1 Create a new Assign activity in the process.
- 2 Open the BPEL Mapper.
- 3 In the target tree on the right, find the partner link to which you want to deliver a new concrete part.
- 4 In the source tree, find a variable containing the new endpoint address.

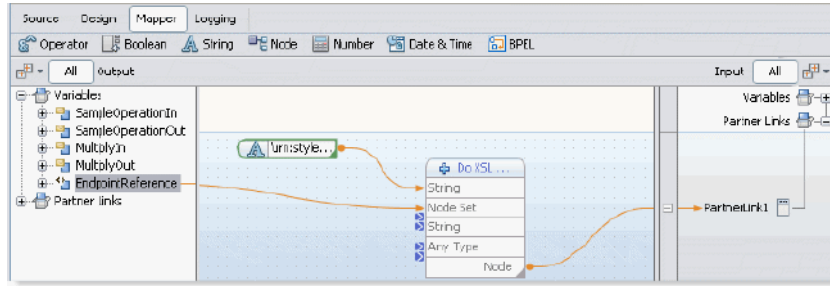
The address of the web-service can be defined in terms of different schemas, and the JBI container requires a special data type called ServiceRefType which is a simple wrapper for any endpoint-describing data type.

To wrap your data:

- In the mapper toolbar, choose BPEL → Wrap with Service Reference.



- This function is a doXslTranform function that uses a predefined XSL-style sheet. A new concrete part is assigned to the partner link.



Note – The runtime supports only schemas included into WS-BPEL 2.0 specification. The WS-Addressing schema is not included in the BPEL specification and as a result it is not supported by the BPEL runtime. When the WS-Addressing schema is used for the first time it is copied from NetBeans global catalog to the BPEL Module project source root and further the project refers to the local copy of the schema. The addressing.xsd schema also appears among the Module's process files in the Projects window.

The Basic Activities

The Basic Activities section of the BPEL Designer Palette contains icons for the following business process elements:

- “Using the Assign Element” on page 66
- “Using the JavaScript Element” on page 67
- “Using the Validate Element” on page 69
- “Using the Empty Element” on page 70
- “Using the Wait Element” on page 70
- “Using the Throw Element” on page 71
- “Using the Rethrow Element” on page 72
- “Using the Exit Element” on page 72
- “Using the Compensate Element” on page 72
- “Using the CompensateScope Element” on page 73

Using the Assign Element

The Assign activity assigns values to variables. You use the Assign element to copy data from one variable to another, construct and calculate the values of expressions, and store new data in variables. Expressions are required to perform simple computation or operate message

selections, properties, and literal constants to produce a new value for variables. The Assign activity can contain one or more elementary assignments.

Usage

Use the BPEL Mapper to define the copy rules for the Assign activity or add expressions. For more information, refer to the “[Assign Activity Scenario](#)” on page 90 section of the guide.

Double-click the Assign activity on the diagram to open Mapper view of the activity. If this window is not visible, you can invoke it manually, by choosing Window → Other → BPEL Mapper from the main menu.

Assign Element Properties

The Properties window of the Assign element, invoked by right-clicking the element and choosing Properties, contains two properties:

- **Name:** The name of the element.
- **Assignments Count:** The number of assignment rules specified for the element.
- **Validate:** Specifies whether the Assign activity validates variables modified by the activity. When the value is “yes”, the Assign activity will validate all of the variables it modifies. If the validation fails because one of the variables is invalid to its corresponding XML definition, then a standard fault `bpel:invalidVariables` is thrown. The default value is N/A.
- **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

Using the JavaScript Element

The JavaScript activity acts like an Assign activity that enables you to use JavaScript (E4X), rather than using XPath 1.0. In many cases JavaScript can perform the same logic as XPath with much fewer processes.

Usage

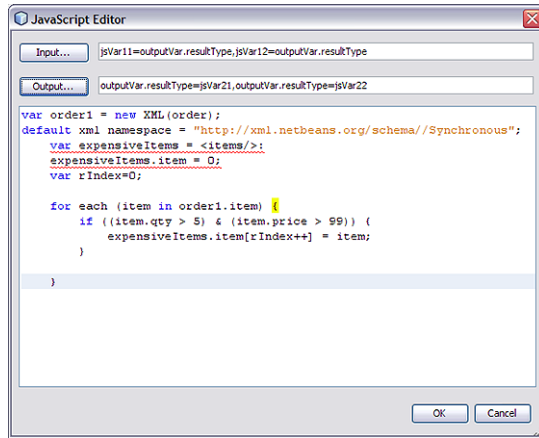
Drag the JavaScript element from the Palette to the diagram. The JavaScript activity is added to the diagram and an empty extension assign operation is added to the source.

```

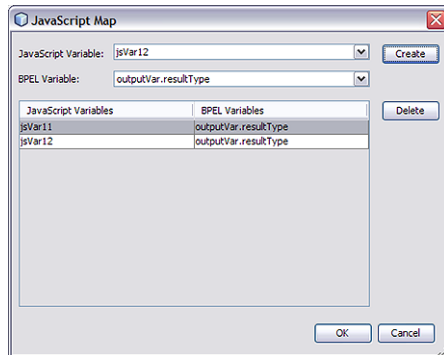
<assign name="JavaScript1">
  <extensionAssignOperation>
    <Expression>
      xmlns="http://www.sun.com/wbpe1/2.0/process/executable/SUNEExtension/DataHandling"
      expressionLanguage="urn:sun:bpe1:JavaScript"
      inputVars="" outputVars=""><![CDATA[
      ]>
    </Expression>
  </extensionAssignOperation>
</assign>

```

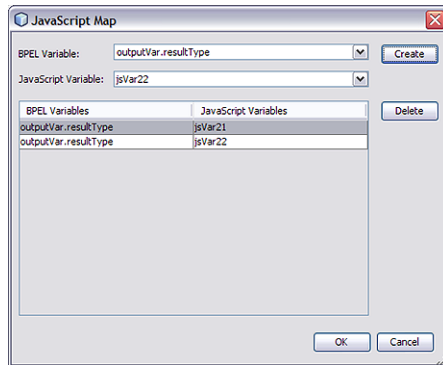
Double-click the JavaScript element in the diagram to open the JavaScript Editor. From the editor you can assign input and output expressions, and add your JavaScript code to the editor's text field.



The JavaScript Editor's Input button opens a dialog box to specify the input mapping between BPEL and JavaScript variables.



In the same way, the editor's Output button opens another dialog box to specify the output mapping between JavaScript and BPEL variables.



JavaScript Element Properties

The Properties window of the Assign element, invoked by right-clicking the element and choosing Properties, contains two properties:

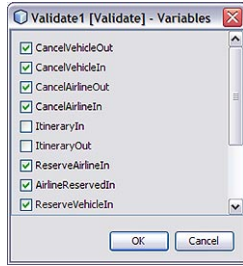
- **Name:** The name of the element.
- **Validate:** Specifies whether the JavaScript (Assign) activity validates any variables modified by the activity. When the value is “yes”, the activity will validate all of the variables it modifies. If the validation fails because one of the variables is invalid to its corresponding XML definition, then a standard fault `bpel:invalidVariables` is thrown. The default value is N/A.
- **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

Using the Validate Element

The Validate element is used to validate the values of variables against their associated XML and WSDL data definition. The element includes a Variables property that lists the variables for the process, and allows you to specify which variables to validate. When one or more variables prove invalid against a corresponding XML definition, a standard fault, `bpel:invalidVariables`, is thrown.

Usage

Drag and drop the Validate element to the diagram between Process Start and Process End. From the Validate Properties, click the Variables edit button. Select the variables you want validated from the Variables dialog box.



Using the Empty Element

The Empty element has no operation associated with it. It is usually used as a placeholder within a process, to catch and suppress faults, or to help synchronize actions within a flow activity that are executed concurrently.

The Empty element can be used when someone else will be implementing a business process, or when the activities within a flow activity need to be synchronized.

Usage

Drag the Empty element from the Palette to the diagram.

Using the Wait Element

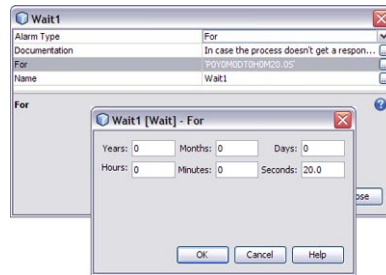
Use a Wait element to specify a wait condition based on a unit of time or a duration. Double-click the Wait element on the diagram to open Mapper view of the activity.

Usage

Drag the Wait element from the Palette to the diagram. Like other elements, it must be placed in the correct position in the process flow; otherwise you will not see the element in the diagram.

Right-click the element in the diagram and choose Properties to invoke a Properties window. Using the Properties window, you can specify:

- **Name:** Specifies a name of the element.
- **Alarm Type:** Specifies the alarm type. The available options are:
 - **For:** Specifies a duration for the process to wait
 - **Until:** Specifies the time until which the process is delayed.
- **For:** Specifies the wait time in years, months, days, hours, minutes, and seconds. Click the edit button to open the Wait For dialog box where you specify the time in accordance with the selected expiration type. * __Documentation__: User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.



- **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

Using the Throw Element

Use this activity to signal an internal fault.

Usage

In defining the properties of this element, you can specify a fault name and a fault variable. These details can then be passed onto a fault handler that is configured to deal with this kind of exception.

Throw Element Properties

The properties of the Throw element are configured from the Properties window.

The options are:

- **Name:** The name of the element.
- **Fault Name:** The Fault Name is a QName. You can use any name. The dialog box lets you choose a standard or already defined fault or specify a new one. The next time the dialog box is opened, the newly defined faults appear inside of User Defined Faults folder. These are collected from the current BPEL file only. Clicking the ellipsis button (...) invokes the Fault Name dialog box.

You can use the WSDL Editor to add fault definitions to the WSDL file. For more information, see the Configuring Port Types Using the WSDL View section of *Using the WSDL Editor*.

- **Fault Variable:** Click the ellipsis button to specify the name of the variable, already declared in the BPEL file, that will contain the fault message.

- **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

Using the Rethrow Element

The Rethrow activity can only be used within a fault handler. The Rethrow activity is used to rethrow the fault caught by the fault handler. Before adding the Rethrow element to the BPEL process, you should add a Fault Handler element to the Process or Scope element and add a Catch or Catch All element to the Fault Handler container.

Usage

- If you have not added a Fault Handler container to the diagram, in the Design view right-click the Scope or Process element and choose Add → Fault Handlers.
- Right-click the Fault Handler container and choose Add → Add Catch or Add → Catch All.
- Drag the Rethrow element from the Palette to the diagram and place it inside the Catch or Catch all element of the Fault Handler container.

There are no properties to be defined for the Rethrow element as it rethrows the fault caught by the Fault Handler.

Using the Exit Element

Use this activity to halt the execution of an activity or a process: either within the process, within a structured activity, or within a handler.

Usage

In the Design view, drag the Exit element from the Palette to the diagram.

Note: The BPEL runtime does not support Exit within the Flow and On Alarm elements, or within the On Event child of the Event Handler element.

Using the Compensate Element

The Compensate element is used to start compensation on all inner scopes that have successfully completed. The Compensate activity can only be used within Catch, CatchAll, Compensation Handler or Termination Handler elements.

Before adding Compensate element to the process, be certain to add one of the following to either the Process or Scope element:

- A Fault Handler and Catch or Catch All
- A Compensation Handler
- A Termination Handler

The Compensate activity causes compensation of all scopes immediately enclosed in the scope containing the fault handler, compensation handler, or termination handler with the Compensate activity.

Usage

From the Palette, drag the Compensate activity and place it inside the Catch, CatchAll, Compensation Handler or Termination Handler element on the diagram. The Compensate activity requires no property configuration, its behavior is predefined.

Using the CompensateScope Element

The CompensateScope element is used to start compensation on a specific scope that has successfully completed. The CompensateScope activity can only be used within Catch, CatchAll, Compensation Handler or Termination Handler elements.

Before adding the CompositeScope element, add one of the following to the Process or Scope element:

- A Fault Handler and Catch or Catch All
- A Compensation Handler
- A Termination Handler

The Compensate Scope activity enables compensation for one specified Scope or Invoke element enclosed into the scope that contains the handler with the Compensate Scope activity by invoking the compensation handler of the Scope or Invoke element.

Usage

1. From the Palette, drag the CompensateScope activity and place it inside the Catch, CatchAll, Compensation Handler or Termination Handler element on the diagram.
2. Right-click the CompensateScope element and choose Properties.
3. In the Properties dialog box, configure the following:
 - **Name:** Enter a name for the element.
 - **Target:** From the drop-down list, select a scope or an invoke activity to be compensated.
 - **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

The Structured Activities

The Structured Activities section of the BPEL Designer Palette contains icons for the following business process elements:

- “Using the If Element” on page 74
 - “Using the While Element” on page 75
 - “Using the Repeat Until Element” on page 76
 - “Using the For Each Element” on page 76
 - “Using the Pick Element” on page 77
 - “Using the Flow Element” on page 78
 - “Using the Sequence Element” on page 79
 - “Using the Scope Element” on page 80

Using the If Element

The If activity supports conditional behavior of a business process instance. The If activity consists of conditional branches defined by the If and Else If elements, followed by an optional Else branch. The conditions on If and Else If branches are evaluated in the order they appear. During execution, the first branch whose condition holds true is taken and provides the activity specified for the If activity. In other words, if there are several Else If branches whose conditions hold true, only the first of them will be executed.

If none of the branches evaluates to true, then the Else path is chosen. If the Else branch is not explicitly specified, this branch is considered to contain an Empty activity. The If activity is complete when the activity of the selected branch completes.

Usage

1. In the Design view, drag the If element from the Palette to the diagram.
2. Double-click the If element on the diagram or select the Mapper tab on the toolbar. The BPEL Mapper opens.
3. Specify the condition for the If element using the BPEL Mapper. For more information, refer to the “[If Activity Scenario](#)” on page 91 section of the guide. You can also specify the condition manually in the Properties window, invoked by right-clicking the element and choosing Properties.
4. (Optional) In the Properties window, enter the name for the If element.
5. Add the element that will be executed if the condition is true into the If element. Configure the nested element. If you add another element into the If element, the nested elements are automatically wrapped in the Sequence element.

6. Add other branches (Else If and Else) as described below.

Adding an Else If Branch to the If Element

1. Right-click the If element and choose Add Else If.
2. Add an activity to the Else If that will be executed if the condition defined for this Else If element is true. To define a condition, use the BPEL Mapper.
3. (Optional) Add more Else If activities by choosing Add Else If and add activities to them.

Adding an Else Branch to the If Element

Drag the activity you want to be executed on the Else branch onto the connector path marked with a slash mark. Configure the nested activity.

Reordering Else If Branches

In the Design view, drag the Else If branch that you want reordered and drop it onto the placeholder that appears next to another Else If branch.

Using the While Element

Use the While element to repeatedly execute one or more activities as long as specific conditions are in place at the beginning of each iteration. This element contains other elements that are repeated while success criteria you specify are met. If the condition you specify leads to false, none of the activities listed will be executed.

Note: the While element first checks the validity of the condition and then executes the iterative activity. Conversely, the [“Using the Repeat Until Element” on page 76](#) element first executes the activity and then checks the validity of the condition.

Usage

1. In the Design view, drag the While element from the Palette to the diagram.
2. Drag an activity that will be repeatedly executed and place it inside the While element. If needed, configure the activity's properties.
3. Use the Properties window to specify the name and condition of the While element. You can enter the condition manually or use the BPEL Mapper to generate the condition for you.

Using the Repeat Until Element

Use the Repeat Until element to repeatedly execute one or more activities as long as specific conditions are in place after the execution of each iteration. This element contains other elements that are repeated until the success criteria you specify are met. If the condition you specify leads to true, the activities listed will be executed once.

Note – the Repeat Until element first executes the iterative activity and then checks the validity of the condition. Conversely, the [While](#) element first checks the validity of the condition and then executes the activity.

Usage

1. In the Design view, drag the Repeat Until element from the Palette to the diagram.
2. Drag activities that will be repeatedly executed and place them inside the Repeat Until element. If needed, configure the activity's properties.
3. Use the Properties window to specify the name and condition of the Repeat Until element. You can enter the condition manually or use the [BPEL Mapper](#) to generate the condition for you.

Using the For Each Element

Use the For Each element to repeatedly execute its contained scope activity exactly N+1 times where N equals the Final Counter Value minus the Start Counter Value.

Usage

1. In the Design view, drag the For Each element from the Palette to the diagram.
2. Add elements that will be repeatedly executed from the Palette into the For Each element. The elements that you add are automatically wrapped into the Scope element.
3. Right-click the For Each element and choose Properties to open its Properties window.

The Properties window for the For Each element includes the following properties:

- **Name:** Specifies the name of the For Each element.
- **Counter Variable Name:** Declares the counter variable name.
- **Start Counter Value:** Sets the start counter value. Use the [BPEL Mapper](#) to generate an integer value expression.
- **Final Counter Value:** Sets the final counter value. Use the [BPEL Mapper](#) to generate an integer value expression.

When the For Each activity is started, the expressions in Start Counter Value and Final Counter Value are evaluated for the first and only time. That is, once the two values are returned they remain constant for the lifespan of the activity. If the Start Counter Value is greater than the Final Counter Value, then no iteration will be performed.

- **Completion Condition:** (Optional) Specifies an integer value expression. After execution of each directly enclosed activity, the number of completed activities is checked against this value. When the number of completed activities equals the value of the specified expression, no further activities are started. When the expression value is greater than the available number of iterations, then no iteration will be started.
- **Count Completed Branches Only:** (Optional) If this checkbox is selected, it tells the runtime to only count the branches that have completed successfully. If this checkbox is cleared, all branches, completed successfully or unsuccessfully, will be counted.
- **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

Using the Pick Element

The Pick element blocks the process and waits until one of the specified events occurs. After the specific event occurs, the activity associated with this event is performed. The possible events are the arrival of a message or a timer-based alarm. The occurrence of the events is mutually exclusive. If more than one of the events occurs, then the selection of the activity to perform depends on which event occurred first.

The Pick activity provides two branches, On Message and On Alarm. The branch whose condition is satisfied first (i.e. a message is received or the specified timer expires) is executed. When you add a Pick element to your diagram, it automatically includes one On Message statement in which you specify the properties of the message that the process awaits from a partner service. Each Pick element must include at least one On Message statement. The On Alarm branch contains a timer you can use to specify how long the process is to wait.

Usage

1. In the Design view, drag the Pick element from the Palette to the diagram.
2. For the On Message branch, configure the properties of the message for which the process is waiting. The configuration is similar to that of the “Usage” on page 57 element.
3. From the Palette, drag the activity that will be executed and place it inside the On Message branch. Configure the activity's properties.
4. (Optional) Add more On Message branches by choosing Add → On Message from the pop-up menu and configure them as described above.
5. (Optional) Add one or more On Alarm branches following the procedure below.

Adding an On Alarm branch

1. Right-click the Pick element and choose Add → On Alarm from the pop-up menu.
2. Configure the timer via the Properties window, invoked by right-clicking the element and choosing Properties. The available options are:
 - **Alarm Type:** Specifies the type of alarm. The type can be one of the following:
 - **For:** Specifies a duration for the process to wait.
 - **Until:** Specifies a deadline for the process.
 - **For/Until:** Used to configure the deadline or the duration for the chosen alarm type. Click the ellipsis button (...) to specify the time. You can also use the [BPEL Mapper](#).
 - **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.
3. Find the activity you want executed after the time expires, and drag it from the Palette to the placeholder inside the On Alarm element.
4. (Optional) Add one or more On Alarm branches as described above.

Pick Element Properties

The Properties window for the Pick element, invoked by right-clicking the element and choosing Properties, includes the following fields:

- **Create Instance:** If set to yes, a new process instance will be created when the specified event occurs. If you do not plan to start a new process instance, keep the default N/A value.
- **Name:** Specifies the name of the For Each element.
- **Documentation:** User documentation attached to the element. This documentation is included with the source code of the BPEL process and can be extracted and included in a report.

Using the Flow Element

Use the Flow element to define a set of activities that will execute concurrently (in parallel).

The Flow activity is a structured activity, containing other activities separated into individual control paths or branches. You can embed as many paths in the activity as you want, and they will all be executed simultaneously.

During execution, each path is executed concurrently, and the activities on each are executed in the order in which they appear, unless they are the source of a link. When the activities are the source of a link, the condition of the link and the join condition of the activity must be evaluated. If the link conditions that lead to an activity conflict with those of its join condition, then a fault is thrown on that activity.

Usage

In the Design view, drag the Flow Element from the Palette to the diagram.

Drag an element to the placeholder inside the Flow element. If you add another element into the same branch of the Flow element, the elements within a branch are automatically wrapped in the Sequence element.

Adding Branches to the Flow Element

You can add one or more branches to the Flow element. The Flow element has a special user interaction style. It always shows a placeholder for the next branch that you might wish to add. To add a new branch, drag an element from the Palette to the immediately available "next branch" placeholder.

Changing the Order of Elements inside Flow

To change the order of activities inside the Flow element:

1. In the Design view, right-click the Flow element and select Change Order.
2. Select an element and use the Move up or Move down buttons to change the position of the element inside the container.

Using the Sequence Element

Use the Sequence element to nest a series of activities into your process. The activities within a sequence will execute in strict sequential order. Process execution returns to the business process when the last activity within the nest has completed.

Usage

Drag the Sequence element from the Palette to the diagram.

Adding Child Activities to the Sequence

You can add one or more child activities to the Sequence. The Sequence element has a special user interaction style. It always shows one or more valid placeholders for the next activity that you might wish to add. To add a new child activity, drag and drop an element from the Palette onto the immediately available next or previous activity placeholder.

Changing the Order of Elements inside Sequence

To change the order of activities inside the Sequence element:

1. In the Design view, right-click the Sequence element and select Change Order.
2. Select an element and use the Move up or Move down buttons to change the position of the element inside the container.

Using the Scope Element

The Scope activity is essentially a collection of child activities that can have their own Variables, Fault and Event Handlers, and correlation sets. The Scope activity provides the behavior context for the child elements. The attributes defined for a parent Scope have local visibility inside this Scope. For example, the variables declared for a Scope are visible only inside that Scope and all nested Scopes. These variables can then be used for the child activities of this Scope.

Usage

1. In the Design view, drag the Scope element from the Palette to the diagram.
2. Right-click the element and choose Add from the pop-up menu to add the following:
 - “Variables” on page 80
 - “Using an Event Handler” on page 105
 - “Using a Fault Handler” on page 103
3. In the Design view, drag elements from the Palette and place them inside the Scope element.
4. Configure the elements.
5. (Optional) Specify the name of the Scope element in the Properties window, invoked by right-clicking the element and choosing Properties.

Variables

Variables in BPEL programming function just as they do in other programming languages: they hold temporary values, form parts of expressions, or are passed as parameters to external partners. Normally, you need a variable for every message sent to or received from a partner service. The BPEL Designer supports the following types of variables:

- **WSDL Message type.** These variables correspond to web service message types that are defined in WSDL files imported by the process. In a BPEL file (.bpe\), these variables must specify a value for the `messageType` attribute. Message type variables are used to hold data in interactions between the process and its partner services.
- **XML Schema type.** These variables correspond to *simple* or *complex* XML Schema data types. The XML schema types themselves are defined in XML Schema files (.xsd) or in WSDL files that are imported into the process. In a BPEL file, variables of this type must specify a value for the `type` attribute.
- **XML Schema element.** These variables correspond to XML Schema elements. The XML schema elements themselves are defined in XML Schema files (.xsd) or in WSDL files that are imported into the process. In a BPEL file, variables of this type must specify a value for the `element` attribute.
- **Built-in type.** Variables of this type are standard simple types defined in the XML Schema specification.

Global and Local Variables

The variables defined at the Process root are global variables, which have a global visibility throughout the entire process. The variables defined within a particular Scope are visible only inside that Scope and all nested Scopes. These variables are called local variables. A variable defined for an inner Scope element can hide an upper defined variable of the same name.

The name of a variable must be unique among the names of all variables defined within the same Scope.

▼ To define a variable:

- 1 **Right-click the Process or Scope element and select Add > Variable.**
- 2 **In the Create New Variable dialog box, name the variable. The name should be unique within this Scope element.**
- 3 **Expand the node corresponding to the type of the new variable and select its type. You have the following options:**
 - **Built-in Types:** Expand the Built-in Types node, select the type's name, and click OK.
 - **Message Type:** Expand a .wsdl file node, select a message type and click OK.
 - **XML Schema:** Expand an .xsd file node or a .wsdl file that contains an embedded schema. Expand the Global Complex Type, Global Simple Type, or Global Elements Simple nodes, select the appropriate type, and click OK.

For your convenience, global types of variables are displayed in bold.
- 4 **(Optional) Clear the Show Imported Files Only checkbox to view the contents of non-imported WSDL and XML schema files.**
- 5 **Click OK.**

By default, the Create New Variable dialog box only shows those files that have already been referenced in the process. However, the project may contain other WSDL and XSD files which have not yet been imported into the process. If you select a type for the new variable that is defined in a non-imported file, the IDE will automatically add the required import to the BPEL process.

You can also add variables from the “[The Navigator Window](#)” on page 29 window. To add a variable, select BPEL Logical View in the Navigator, expand your BPEL Module project's node, right-click the Variables node and choose Add Variable.

▼ **To edit a variable:**

- 1 In the Navigator window, select BPEL Logical View.
- 2 Expand BPEL Module project's node > Variables and double-click the variable you want to edit.
- 3 In the Property Editor dialog box for the variable, change the variable type and name.
- 4 Click OK.

Using the BPEL Mapper

The section describes how to use the BPEL Mapper.

[“About the BPEL Mapper” on page 82](#)

[“Creating BPEL Mappings” on page 83](#)

[“Working with Predicates” on page 85](#)

[“XPath Function Reference” on page 86](#)

[“Mapping Examples” on page 89](#)

About the BPEL Mapper

The BPEL Mapper provides a framework for processing and directing BPEL process data. This framework consists of the following components:

- **Menu Bar.** The menu bar provides operators, necessary elements, and XPath functions you use to create BPEL mappings. You can also enhance or extend BPEL mappings by incorporating predicates that consist of XPath functions.

On the right side of the menu bar, you can use the following buttons to work with function boxes, or functoids:

- **Expand Mapped Nodes** — Expands all mapped nodes.
- **Collapse All** — Collapses all function boxes. This command is useful when you work with numerous function boxes in the mapping pane.
- **All** — All nodes are displayed.
- **Output** — . Only connected nodes will be displayed.
- **Source tree pane** — The source tree pane is placed on the left and contains a tree component that provides access to a business process's data variables and partnerlinks.

- **Mapping pane** — The mapping pane contains a canvas for creating BPEL mappings. When you select a function from the menu bar, a function box appears in the mapping pane. If the function accepts any arguments, then the left side of the function box has one connector for each argument. If an argument is optional, then a question mark appears after the argument name. The right side of the function box has one connector for the result. You can use the BPEL Mapper with the following business process elements:
 - **Assign activity** — You can define one or more copy assignments.
 - **If activity** — You can define the condition.
 - **Else If element within an If activity** — You can define the condition.
 - **For Each activity** — You can define the condition.
 - **Repeat Until activity** — You can define the condition.
 - **While activity** — You can define the condition.
 - **Wait activity** — You can specify the deadline or duration.
 - **onAlarm event** — You can specify the deadline or duration.
- **Destination tree pane** — This pane is placed on the right. Tree component of the destination pane depends on the business process element that you are mapping. The pane contains the following components:
 - For an Assign activity, the right pane contains the same tree component as the left pane.
 - For an If activity, Else If element, For Each activity, Repeat Until activity, and While activity, the right pane contains a Result node.
 - For the Wait activity, the right pane contains a Deadline or Duration node.

▼ To open the BPEL Mapper window:

- Open the BPEL diagram and do one of the following:
 - a. Double-click the element that requires BPEL Mapper. The Mapper tab opens.
 - b. Select an element that requires the mapper and on the diagram toolbar click the Mapper tab.

If you want to see the mapper and the diagram of the process at the same time, you can place the mapper to a separate window. For more information see Cloning Document Views.

Creating BPEL Mappings

You can create a mapping from the source tree pane directly to the destination tree pane, without using any of the functions. This type of mapping can be any of the following:

- Variable to variable
- Part to part
- XSD element to XSD element

- XSD attribute to XSD attribute
- Partner link to partner link

You can also create a mapping that uses one or more [XPath functions](#) from the BPEL Mapper's menu bar. For example, if the BPEL process includes a Wait activity that waits for a period of time, then you can use the Duration Literal function to specify the duration.

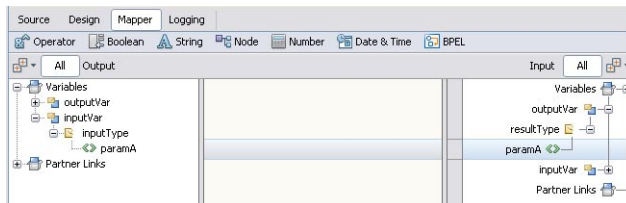
▼ To create a mapping without using any functions:

- 1 In the source tree pane, expand the tree component until the node that you want to map from appears.
- 2 If the destination tree pane contains a tree component, then expand the tree component until the node that you want to map to appears.
- 3 Select the node in the source tree pane and drag the pointer to the node in the destination tree pane.

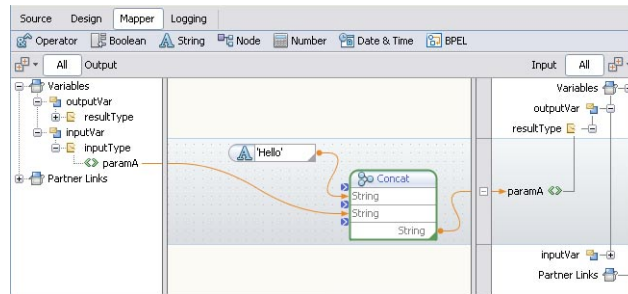
A link connects the nodes.

▼ To use a function in a mapping:

- 1 In the destination tree pane, expand the tree and select the node you want to map to. A blue area appears on the mapping pane. The functions you choose will appear here.



- 2 Click the drop-down menu that contains the function.
- 3 Click the function.
A function box appears in the mapping pane.
- 4 Map any arguments into the appropriate connector on the left side of the function box. The source can be a node in the source tree pane or the output from another function box. If an argument is optional, then a question mark appears after the argument name.
- 5 Map the result from the right side of the function box. The destination can be a node in the destination tree pane or the input into another function box.



▼ To delete a link or function in a mapping:

- 1 Select the link or function.
- 2 Press Delete.

Working with Predicates

The BPEL Mapper enables you to create a predicate that consists of XPath functions.

A predicate applies a condition to a node that can have multiple values. The result is the subset of nodes that satisfy the condition.

For example, assume that a node represents the number of products. If you want to select all products whose number is greater than 50, then you can use the `greater than` and `number literal` functions to define the condition.

Once you create a predicate, you can use the predicate in an assignment. For example, you can copy data from a predicate in the source tree pane to a node in the destination tree pane.

You can edit or delete an existing predicate.

▼ To create a predicate:

- 1 In the source tree pane, right-click a node (schema element or attribute) and choose **New Predicate**.
The Predicate Editor appears.
- 2 Use XPath functions to create the condition for the predicate. Map the result to the predicate node in the destination tree pane.

3 Click OK.

The editor adds the predicate node immediately below the original node. The condition appears in brackets.

▼ **To edit a predicate:**

1 In the source tree pane, right-click the predicate node and choose Edit Predicate.

2 Modify the condition.

3 Click OK.

▼ **To delete a predicate:**

1 In the source tree pane, right-click the predicate node and choose Delete Predicate.

2 Click Yes.

XPath Function Reference

A collection of XPath functions are available in the BPEL Mapper's menu bar. These functions are based on the XPath 1.0 specification.

Each function has zero or more arguments. Each function returns a single result.

The menu bar contains the following drop-down menus: “Operator” on page 86, “Boolean” on page 87, “String” on page 87, “Nodes” on page 88, “Number” on page 88, “Date & Time” on page 89, and “BPEL” on page 89.

Operator

The Operator menu contains the following functions:

- **Greater** — Greater than.
- **Greater or Equal**
- **Less** — Less than.
- **Less or Equal**
- **Addition**
- **Subtraction**
- **Multiplication**
- **Division** — The operator returns the quotient for a given dividend and divisor.

- **Remainder** — The operator returns the remainder for a given dividend and divisor #
Negative
- **Negative**
- **Not Equal**
- **Equal**

Boolean

The Boolean menu contains the following functions:

- **Logical And** — If both arguments are true, then the function returns true. If either argument is false, then the function returns false.
- **Logical Or** — If either argument is true, then the function returns true. If both arguments are false, then the function returns false.
- **Logical Not** — If the argument is false, then the function returns true. If the argument is true, then the function returns false.
- **Language** — Returns true or false depending on whether the language of the context node is the same as or is a sublanguage of the language specified in the argument.
- **Logical False** — Returns false.
- **Logical True** — Returns true.
- **Boolean** — Converts the argument to a boolean. For detailed information about the logic, see the XPath 1.0 specification.

String

The String menu contains the following functions:

- **Contains** — Uses the following logic: If the first argument string contains the second argument string, then the function returns true. Otherwise, the function returns false.
- **Normalize Space** — Returns the argument string with whitespace normalized by stripping leading and trailing whitespace and by replacing sequences of whitespace characters with a single space.
- **String** — Converts an object to a string.
- **Starts With** — Uses the following logic: If the first argument string starts with the second argument string, then the function returns true. Otherwise, the function returns false.
- **String Length** — Returns the number of characters in the string.
- **Substring** — Returns the substring of the first argument starting at the position specified in the second argument with the length specified in the third argument. The position of the first character is 1, the position of the second character is 2, and so on. The third argument is

optional. If the third argument is not specified, then the function returns the substring starting at the position specified in the second argument and continuing to the end of the string.

- **Substring Before** — Returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string. If the first argument string does not contain the second argument string, then the function returns an empty string.
- **Substring After** — Returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string. If the first argument string does not contain the second argument string, then the function returns an empty string.
- **Translate** — 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.
- **Concat** — Returns the concatenation of the arguments.
- **String Literal** — Enables you to enter a string literal.

Nodes

The Nodes menu contains the following functions:

- **Local Name** — Returns the local part of the expanded name of the node in the argument node-set that is first in document order. (An expanded name consists of a local part and a namespace URI.)
- **Name** — Returns the qualified name that represents the expanded name of the node in the argument node-set that is first in document order. (An expanded name consists of a local part and a namespace URI.)
- **Namespace URI** — Returns the namespace URI of the expanded name of the node in the argument node-set that is first in document order. (An expanded name consists of a local part and a namespace URI.)
- **Position** — Returns the context position.
- **Last** — Returns the context size.
- **Count** — Returns the number of nodes in the argument node-set.

Number

The Number menu contains the following functions:

- **Number** — Converts the argument to a number. For detailed information about the logic, see the XPath 1.0 specification.
- **Number Literal** — Enables you to enter a number literal.
- **Round** — Returns the number that is closest to the argument and that is an integer.

- **Sum** — 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.
- **Floor** — Returns the largest number that is not greater than the argument and that is an integer.
- **Ceiling** — Returns the smallest number that is not less than the argument and that is an integer.

Date & Time

The Date & Time menu contains the following functions:

- **Current Date** — Provides the current date.
- **Current Time** — Provides the current time.
- **Current Date and Time** — Provides the current date and time.
- **Deadline** — Provides the specified end time.
- **Duration Literal** — Enables you to enter a duration literal. Use the format specified in the XML Schema specification.

BPEL

The BPEL menu contains the following functions:

- **Do XSL Transform** — This is an XPath extension function defined in WS-BPEL specification. It can be used in a BPEL Assign activity to call an XSLT transformation.
- **Get Variable Property** — Provides the property of the variable.
- **Wrap with Service Reference** — This is a special case of Do XSL Transform function used to transform data into Service
- **Do Marshal** — Performs serialization of an object.
- **Do UnMarshal** — Performs deserialization of an object.
- **GUID** — Provides the GUID Name.
- **Get BP ID** — Gets the Business Process Identification.
- **XML Literal** — Enables you to enter an XML literal.

Mapping Examples

These examples illustrate several mapping scenarios:

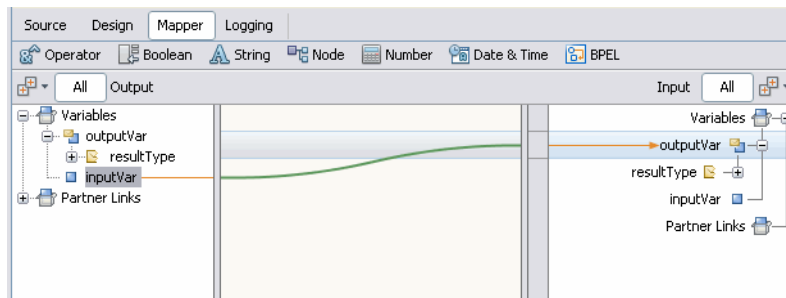
- [“Assign Activity Scenario” on page 90](#)
- [“If Activity Scenario” on page 91](#)
- [“Predicate Scenario” on page 91](#)

Assign Activity Scenario

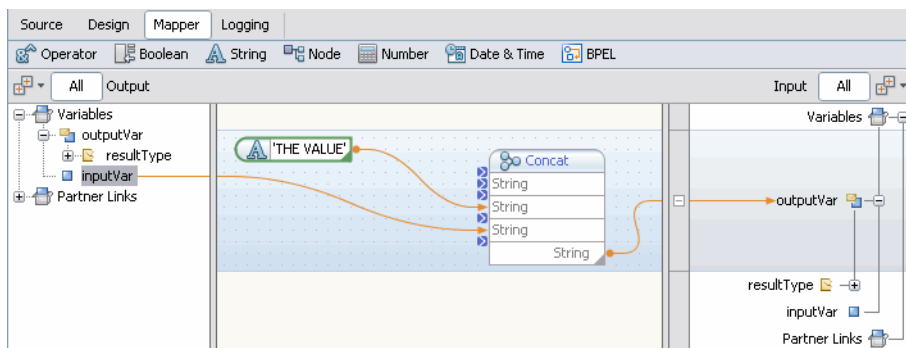
Assume that you want a BPEL process to copy data received from a partner. Do the following tasks:

1. Add an Assign activity after the Receive activity.
2. Use the BPEL Mapper to define one or more copy assignments. To open the BPEL Mapper, double-click the Assign activity on the diagram or select the Mapper tab on the toolbar.

The following example shows a copy assignment that does not use any XPath functions. The itinerary part of the ItineraryIn variable is copied to the itinerary part of the ItineraryOut variable. Notice that the left pane and the right pane contain the same tree component.



The following example shows a copy assignment that uses the concat XPath function. The input variable paramA is concatenated to the end of the string literal Parameter A: and copied to the output variable paramA.

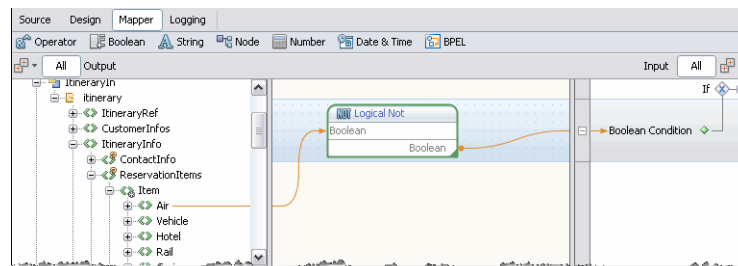


If Activity Scenario

Assume that you want to execute a series of steps only if a certain condition is true. Do the following tasks:

1. Add an If activity to the BPEL process.
2. Use the BPEL Mapper to define the Boolean condition. To open the BPEL Mapper, double-click the If activity on the diagram or select the Mapper tab on the toolbar.
3. Add the steps inside the If activity.

The following example shows a mapping for the condition. The mapping uses the `Not XPath` function, which is available from the `Boolean` node on the menu bar. If the itinerary has an airline reservation, then the `Not XPath` function returns true. The result is mapped to the `Result` node in the right pane.

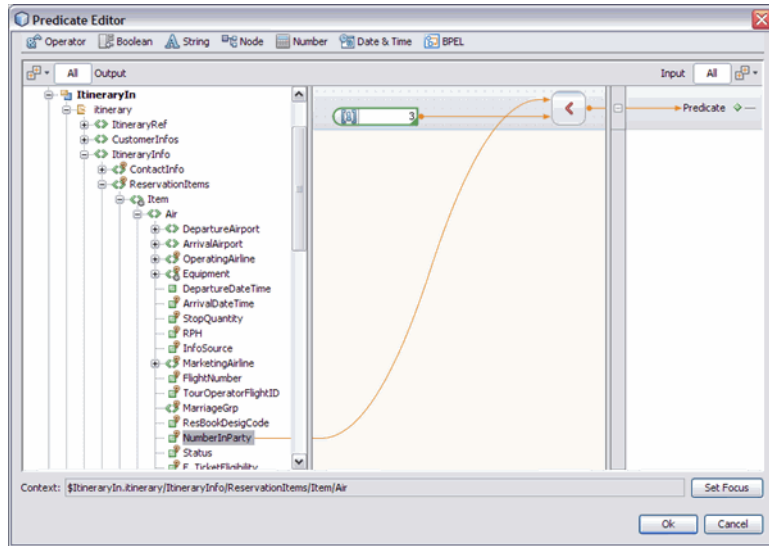


Predicate Scenario

Assume that you want a BPEL process to copy itinerary data from itineraries of customers with no more than two in their party. The input records include a variable that specifies the number of passengers in the customer's party. Do the following tasks:

1. In the left pane of the BPEL Mapper window, right-click the repeating node that is marked with an asterisk (*) and choose `New Predicate`.
The Predicate Editor window appears.
2. Add the `Less XPath` function to the middle pane.
3. Add the `number literal XPath` function to the middle pane. Set the value to 3.
4. Map the variable node to the first argument of the `Less XPath` function.
5. Map the result of the `number literal XPath` function to the second argument of the `Less XPath` function.
6. Map the result of the `Less Than XPath` function to the `Result` node in the right pane.
7. Click `OK`.

The following example shows how the mapping appears in the Predicate window. Once you click OK, you can use the predicate node in a copy assignment.



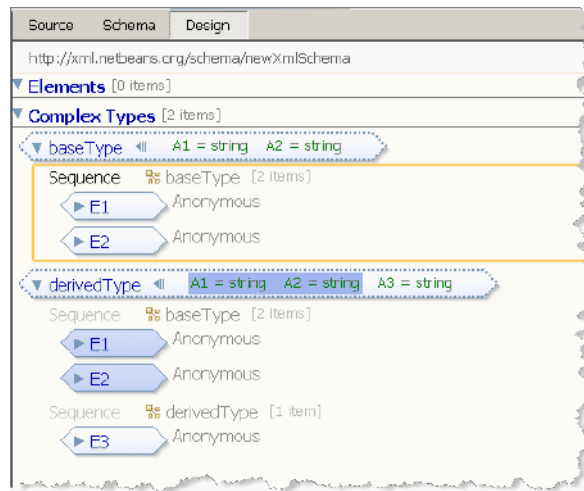
If the BPEL process received the following XML, then the predicate would select the first `Air` tag.

```
<Air>
  <NumberInParty>2</NumberInParty>
</Air>
<Air>
  <NumberInParty>4</NumberInParty>
</Air>
<Air>
  <NumberInParty>6</NumberInParty>
</Air>
```

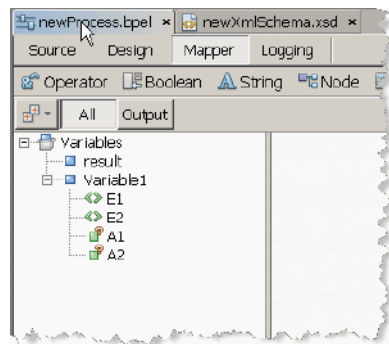
Using Type Cast and Pseudo-Components

Type Cast and Pseudo-components are provided to support type inheritance, help build message structure, and prevent validation errors. Type Casting addresses a problem in which a base complex type does not contain the same elements as the derived type.

For example, a variable may have a base type, containing two attributes and two elements, and a derived type inherited from the base type, containing an additional attribute and element.



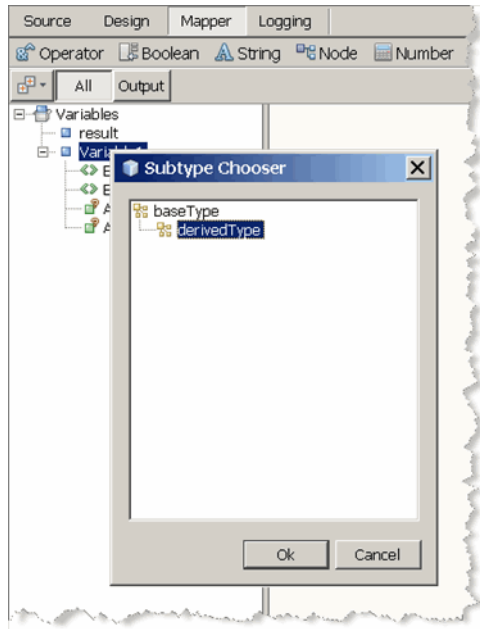
When the message is declared in a WSDL for use in BPEL, the base type is used to declare the message. So, in this example, from the Mapper view of the XML structures we see the variable's base type that does not contain the additional element and attribute.



Since the Mapper is designed to allow you to graphically link elements and attributes, these structures need to be available in the mapper view. To allow mapping of derived type attributes and elements, which are implied by the base type, the Mapper enables you to cast an object to another type - in this example, to the derived type.

To cast an object to another type:

1. From the Mapper view of the project's bpel file, right-click the variable that has the base type and click Cast To in the pop-up menu. The Subtype Chooser appears displaying a tree with the original type as the root and subtypes under the root. If there are no subtypes, only the original type will appear in the box.



2. To cast the original variable to the derived type, select derivedType in the Subtype Chooser and click OK. The derivedType variable appears in the Mapper.
3. Expand the derivedType variable in the Mapper and note that the additional attribute and element are now available to use for mapping.

Type Cast

The Type Cast option allows you to explicitly cast an object to another type. This only works for objects that have a schema type.

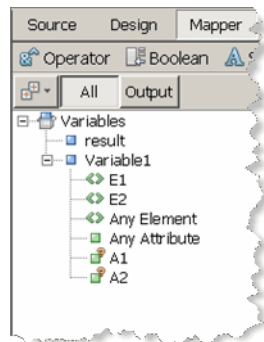
In the BPEL Mapper the following object can be casted:

- Schema Element
- Schema Attribute
- Variable (except variables of the WSDL Message type)
- Message Part

Pseudo-Component

The Pseudo-component feature is similar to Type Cast. The XML Schema introduces `xsd:any`. It declares that an element can be located in the XML document and have any name and type. The BPEL mapper allows you to add to the tree elements, which correspond to `xsd:any`. Such elements don't appear in the schema, but they look similar to other elements. That is why they are called pseudo-elements. The user can map from and to a pseudo-element the same way as for an ordinary element. This also applies to the `xsd:anyAttribute`. The user can create pseudo-attributes to use for mapping and design purposes.

In the Mapper's Input and Output trees, `xsd:any` appears as an element node with the name "Any Element" and `xsd:anyAttribute` as an attribute with the name "Any Attribute". Both have the pop-up menu item "Cast To...", which calls the special dialog box used to specify a name, namespace and type for the new pseudo-component. For `xsd:any` any global type can be chosen. For `xsd:anyAttribute` only the global simple types are suggested. In both cases the settings allow you to choose global types from several different sources. The completed pseudo-components appear in the tree with the name pattern `(castToElementName)Any Element` or `(castToAttributeName)Any Attribute`.



Type Cast and Validation

Validation is passive in regard to type casts and pseudo-components, meaning that validation does not object to an unknown component if it is a qualified type cast or it is declared as a pseudo-component.

Type Cast and Pseudo Component Limitations

The type cast is used to avoid runtime errors, but there are several reasonable limitations:

- An object of a specific schema type can be cast only to a subtype. This applies to elements, attributes, variables and message parts. The XML Schema has two derivations, extension and restriction, both of which are supported by type cast.
- `xsd:any` contains additional attributes which might restrict the possible element type.

- Only global types can be used when you declare a pseudo-component. Global attributes or elements cannot be used. The target of the type must be referencable, and as such should be global.
- You cannot nest type casts, pseudo-components, and mixed declarations.
- You cannot add a new predicate to a casted component or its part.
- You cannot mix two or more type casts or pseudo components in an XPath expression. A type cast cannot be associated with a specific part of an expression, but only with the expression as a whole.

Using Normalized Message Properties

Normalized Message properties are commonly used to specify metadata that is associated with message content. `javax.jbi.security.subject` and `javax.jbi.message.protocol.type` are two examples of standard normalized Message properties defined in the JBI Specification.

Normalized Message properties are used to provide additional capabilities in Open ESB, such as:

- Getting and Setting transport context properties. For example, HTTP headers in the incoming HTTP request, or file names read by the File Binding Component
- Getting and Setting protocol specific headers or context properties (SOAP headers)
- Getting and Setting additional message metadata. For example, a unique message identifier, or an endpoint name associated with a message
- Dynamic configurations. For example, to dynamically overwrite the statically configured destination file name at runtime

Some of the use cases mentioned above require protocol/binding specific properties, typically used by a particular binding component. Other properties are considered common or general purpose properties that all participating JBI components make use of, for example, the message ID property, which can be utilized to uniquely identify or track a given message in the integration.

Using Normalized Message Properties in a BPEL Process

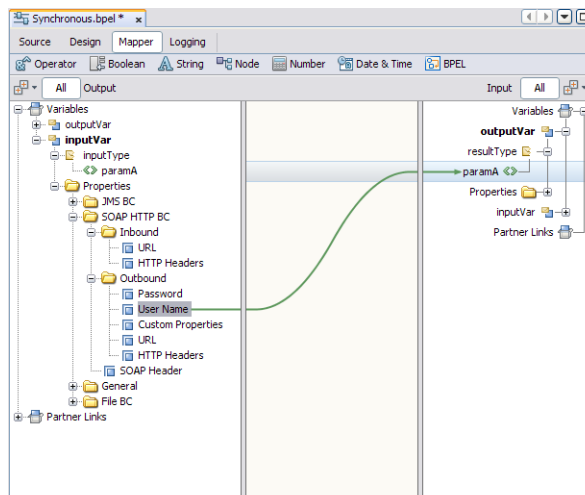
The Normalized Message properties are accessed from the BPEL Designer Mapper view. When you expand a variable's Properties folder it exposes the variables predefined NM properties. If the specific NM property you need is not currently listed, additional NM properties can be added.

Using Predefined Normalized Message Properties in a BPEL Process

Predefined Normalized Message properties are ready for use, from a variable's Properties file.

▼ To use predefined normalized message properties in a BPEL process

- 1 From the Design View diagram, select the activity with the process you want to edit.
- 2 Click **Mapper** to switch to the Mapper view of the BPEL process.
- 3 From the **Output** pane, expand the **Variable** you want to edit and its **Properties** file. The Properties file contains the predefined Normalized Message (NM) properties.



- 4 To use a predefined NM Property, select the property and use it to build an expression or an assign.

Adding Additional Normalized Message Properties to a BPEL Process

If the specific NM Property you want is not listed, you can add additional NM properties.

There are two options available when adding NM Properties:

- **Add NM Property Shortcut:** This option typically supports *simple type* properties, in that it does not grant access to some data within the NM Property.
- **Add NM Property:** This option provides access to data within the NM property used to build expressions.

▼ To add a Normalized Message Property Shortcut to a BPEL process

- 1 From the Output or Input panes of the BPEL Mapper, expand the node for the variable to which you want to add an NM property. Right-click that variables Properties directory node and select Add NM Property Shortcut from the pop-up menu.

The Add NM Property Shortcut dialog box appears.

- 2 Enter the information for the new NM property into the the Add NM Property Shortcut dialog box, as follows:
 - a. Property Name: The NM property name (see each binding component's documentation for available NM properties).
 - b. Display Name: The display name for the NM property. This is a user-defined name that appears in the Mapper tree. The display name is optional.



- 3 Click OK.

The new NM property is added to the Mapper tree under the variables Properties directory. The property can now be used in assigns and to build expressions.

▼ To edit an NM Property Shortcut

- 1 To edit an existing NM property shortcut, right-click the NM property shortcut in the BPEL Mapper tree and choose Edit NM Property Shortcut in the pop-up menu.

The Add NM Property Shortcut dialog box appears.

- 2 Edit the NM Property Name or Display Name, and click OK.

▼ To delete an NM Property Shortcut

- 1 To delete an NM property shortcut, right-click the property in the Mapper tree.
- 2 Choose **Delete NM Property Shortcut** in the pop-up menu.

The NM Property Shortcut is deleted.

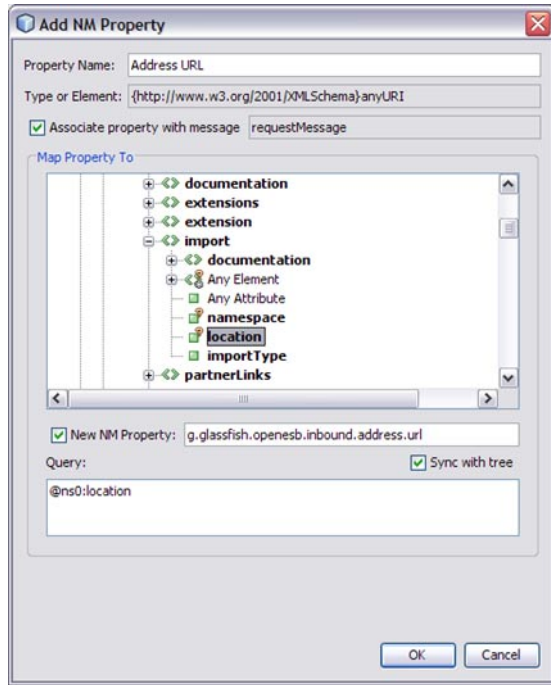
▼ To add a Normalized Message Property to a BPEL process

- 1 From the Output or Input panes of the BPEL Mapper, expand the node for the variable to which you want to add an NM property. Right-click that **variables Properties** directory node and select **Add NM Property** from the pop-up menu.

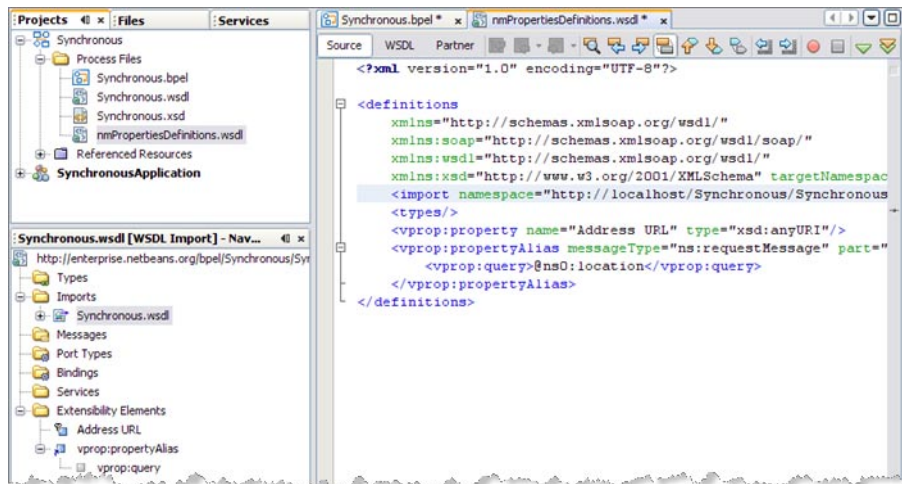
The Add NM Property dialog box appears.

- 2 Enter the information for the new NM property in the the Add NM Property dialog box, as follows:
 - a. **Property Name:** User-defined property name. This name is displayed in mapper tree and stored in WSDL file.
 - b. **Type or Element:** Displays the property type or element associated with the selected node in the Map Property To tree.
 - c. **Associate property with message:** To associate the new NM property with all variables of any message type select this checkbox.
 - d. **Map Property To:** The Map Property To tree displays all of the predefined NM properties. This is used to build a query or choose a property type.

When you select a node within the property tree the Type or Element and Query fields are populated automatically. Valid endpoint nodes are displayed in bold.
 - e. **New NM Property:** Select the **New NM Property** checkbox to add a specific NM property, and enter the name of the property in the **New NM Property** field. The new NM property is added to the Map Property To tree.
 - f. **Sync with tree:** When this checkbox is selected, the Query field is automatically synchronized with the selected node in the Map Property To tree.
 - g. **Query:** Displays the query type associated with the selected node in the Map Property To tree.



- 3 Click OK. The new NM property name is added to the tree in the BPEL Mapper, and the NM property is stored in `nmPropertiesDefinitions.wsdl` as a pair of elements: `<vprop:property>` and `<vprop:propertyAlias>`



The new NM property can now be used in assigns and to build expressions.

▼ To delete an NM Property

- 1 To delete a new NM property, right-click the property in the Mapper tree.
- 2 Choose Delete NM Property in the pop-up menu.

The property is deleted.

BPEL Code Generation Using NM Properties

Data copied from an NM property or an NM property shortcut generates code that is similar to the following:

```
<from variable="inputVar" sxnmp:nmProperty="org.glassfish.openesb.file.outbound.dcom.username"/>
```

Data copied from WSDL properties based on NM property generates code that is similar to the following:

```
<from variable="inputVar" property="ns3:DemoNMPProperty"/>
```

When properties and NM properties are used to build an expression, code similar to the following code is generated:

```
<from>concat(bpws:getVariableProperty('inputVar', 'ns3:DemoNMPProperty'),
sxnmp:getVariableNMPProperty('inputVar', 'org.glassfish.openesb.file.outbound.dcom.username'))</from>
```

An NM property used in a condition generates code that is similar to the following:

```
<condition>sxnmp:getVariableNMPProperty('inputVar', 'my.nmProperty.boolean')</condition>
```

General Normalized Message Properties

Normalized Message properties are either *General*, available to all participating JBI components, or *protocol/binding specific*, used by a particular binding component. The following General NM properties are available to all binding components.

TABLE 1 General Normalized Message Properties

Property Name	Type	Description and Use
org.glassfish.opensb.messaging. groupid	java.lang.String	Uniquely identifies a message with the group to which a message belongs. For example, it applies the RM sequence group number for SOAP messages, or a time stamped file name (where the file record message comes from). This property is optional.
org.glassfish.opensb.messaging. messageid	java.lang.String	Uniquely identifies a message. For batch processing this might be a record number (for example, a particular record in a file), or a GUID. This property is mandatory.
org.glassfish.opensb.messaging. lastrecord	java.lang.String	The value is a string representation of boolean ("true" or "false"). This property can be used to signal the last record in a group, e.g. the last record in a RM sequence for SOAP messages, or the last record in a file when multiple record processing is turned on for File BC. This property is optional.
org.glassfish.opensb.exchange. endpointname	java.lang.String	The value a string representation of the endpoint name set on the exchange. This represents the endpoint name of the "owner" of the message, and could be made available by JBI runtime.

Binding Component Specific Normalized Message Properties

Binding components each have their own protocol specific Normalized Message properties. These include inbound and outbound specific, as well as general NM properties for each binding component.

For a list of binding component specific NM properties, refer to the following:

- [File Binding Component NM Properties](#)
- [HTTP \(SOAP\) Binding Component NM Properties](#)
- [JMS Binding Component NM Properties](#)

Using Handlers

The following sections describe, in order of their appearance:

- The circumstances under which you would use a specific handler.
- The use of those elements within the context of the BPEL Designer.

[“Using a Fault Handler” on page 103](#)

[“Using an Event Handler” on page 105](#)

[“Using a Compensation Handler” on page 108](#)

[“Using a Termination Handler” on page 108](#)

Using a Fault Handler

The BPEL language provides the capability to catch and manage exceptions using fault handlers. For example, exceptions occur when web services return different data than was expected. If faults are not handled, the entire BPEL process can be thrown into a faulted state. Therefore, to prevent the entire process from fault, you can add fault handlers to catch and manage exceptions within particular Scopes.

When to Use

Each fault handler contains an activity that runs in case of an error. For example, a partner service is notified if an error has occurred. Fault handlers can be added to the entire process or to individual Scope elements.

You can attach one Fault Handler container to either the Process or the Scope elements. Inside the Fault Handlers container, you can create several Catch activities configured to catch specific kinds of faults, or one Catch All handler element to catch all the exceptions not caught by specific handlers.

Usage

1. Right-click the Scope or Process element and choose Add > Fault Handlers.
An empty container element appears.
2. Right-click the Fault Handler container and choose Add > Catch or Add > Catch All.
You may add as many specific Catch elements as you wish to the Fault Handlers group. You can add only one Catch All element per Fault Handlers container.
3. Add an activity to the Catch or Catch All element that will be executed in case of a fault.

Catch Element

Use this element to intercept and deal with a specific kind of fault.

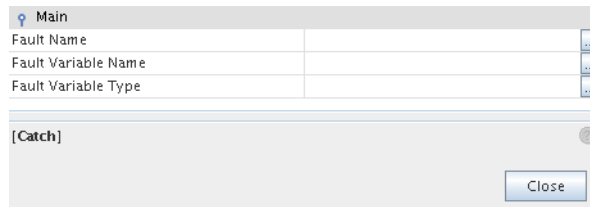
This element is used within an appropriate Fault Handlers container element.

Catch Element Properties

The properties of the Catch element are defined in the Properties window. You can also right-click the element on the diagram and choose Properties.

The available properties are:

- **Fault Name** — Selects the qname of the fault from the list of faults, which contains faults defined in the WSDL files. You can also specify a new user-defined fault in a fault handler or in a throw.
- **Fault Variable Name** — Specifies the name of a variable. The Catch is the receiver of the fault. The fault is produced by either a throw, a partner, or system. The fault variable puts the message's data into the fault. This variable is created and initialized in the Catch.
- **Fault Variable Type** — Specify the type of the variable. The type can be either Message (WSDL) or Global Element (XSD).



Catch All Element

Use the Catch All element to intercept and deal with all faults that are not caught by an associated catch element.

The Catch All element is used within a fault handler window along with one or more Catch elements. It is defined within a Fault Handlers container element along with one or more Catch elements.

There are no properties for the Catch All element. Its behavior is pre-defined and requires no property configuration.

Using an Event Handler

The entire BPEL process as well as each individual Scope can be associated with a set of Event Handlers that are invoked concurrently if the corresponding event occurs. The actions taken within an Event Handler can be any type of activity, such as Sequence or Flow. The only immediate child of an Event Handler is Scope, so when you drag an element from the Palette into an Event Handler, it is automatically wrapped in Scope.

When to Use

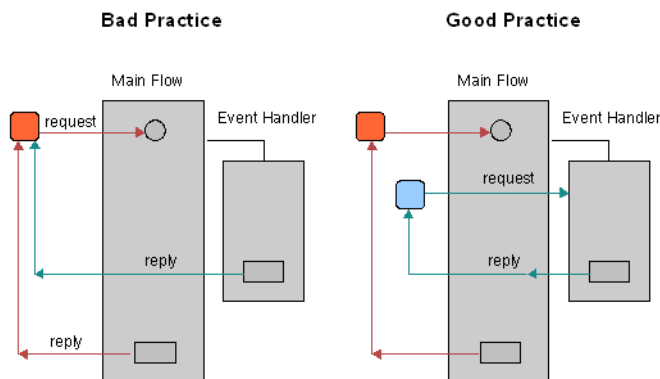
There are two types of events:

- Incoming messages, which correspond to a request/response or one-way operation in WSDL. These messages are specified using the On Event elements.
- Alarms, or timers, which invoke activities after the specified periods or when a deadline is reached. The times are specified using the On Alarm elements.

Note – Event handlers do not interfere with the main flow of the business process. If an event occurs, and an event handler is executed, the main flow will be executed also.

This means, one shouldn't use event handlers, for example, to send replies to requests received in the main flow, because the main flow might also send a reply which will never be seen by the client, and this can lead to various issues.

On the contrary, a good use-case for event handlers would be to request a business process status. In this case, the activities in the event handler will collect data about the state of the business process, and then reply to the request that triggered the event handler.



This picture can be a bit misleading. The request should be caught by handler itself but not by an internal Receive. I mean that the inbound arrow should come to handler itself, but not to an internal circle, which can be treated like a receive.

Usage

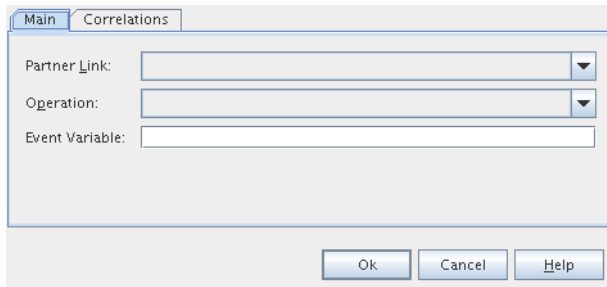
1. Right-click the Process element or any Scope and choose Add → Event Handlers.
2. Once you have added an Event Handlers container, you can right-click on the Event Handlers element to add an On Event or On Alarm branch. You may add as many specific On Event or On Alarm elements as you wish, to the Event Handlers group.

On Event Element

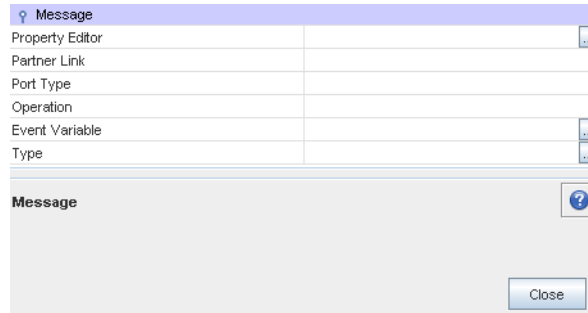
The On Event element indicates that the specified event waits for a message to arrive. The interpretation of this tag and its attributes is very similar to a Receive activity.

Usage

1. Right-click the Event Handlers container and choose Add > On Event.
2. Double-clicking the On Event element opens a Property Editor where you can specify/change the following:
 - The partner link
 - The operation associated with the On Event element
 - The event variable



3. Right-click the On Event element and choose Properties to open a Properties window to review and modify the properties of the element. In addition to the properties present in the Property Editor dialog box, the Properties window contains the Port Type and Type field.



The Correlations tab in the On Event Property Editor dialog box enables you to examine or specify a correlation set.

The tab shows:

- A correlation sets' name
- The initiation of a correlator

For more information see [“Understanding Correlation. Using the Correlation Wizard”](#) on page 109.

On Alarm Element

The On Alarm element specifies the deadline for or the duration of the nested Scope.

On Alarm Element Properties

The properties of the On Alarm element are defined in the Properties window, invoked by right-clicking the element on the diagram and choosing Properties. The available properties are:

- Alarm Type is used to choose the type of alarm. The available options are:
 - **For** — Sets the duration for the process to wait.
 - **Until** — Specifies the deadline for the process.
 - **Repeat Every** — . Specifies the frequency of process initiation. It initiates the process each time the specified duration period expires. The clock for the very first duration starts when the associated scope starts.
 - **For + Repeat Every** — Specifies the frequency of process initiation after the duration of a specified wait time. The process is initiated each time the duration period specified in the Repeat Every field expires. The first alarm is fired when the period of time specified in the For field expires.

- **Until + Repeat Every** — Specifies the frequency of process initiation based on the specified deadline. The process will be initiated each time the duration period specified in the Repeat Every field expires. The first alarm is fired when the deadline specified in the Until field is reached.
- The second (and third, where available) property is used to specify the duration or deadline for the selected alarm type.

Using a Compensation Handler

A business process often contains several nested transactions. The overall business transaction can fail or be cancelled after many enclosed transactions have already been processed. Then it is necessary to reverse the effect obtained during process execution. For example, a travel planning process can include several nested transactions to book a ticket, to reserve a hotel and a car. If the trip is cancelled, the reservation transactions must be compensated for by cancellation transactions in the appropriate order. For such cases, WS-BPEL provides you with the capability to define compensation actions.

When to Use

A Compensation Handler is a container for the activities that perform compensation actions. You can add one Compensation Handler to either the Scope or the Invoke elements. The compensation handler can be invoked by [“Using the CompensateScope Element” on page 73](#) [“Using the Compensate Element” on page 72](#) activity.

▼ To add a Compensation Handler to Scope or Invoke elements:

- 1 **Right-click the Scope or Invoke element and choose Add > Compensation Handler. An empty container element appears.**
- 2 **From the Palette, drag one or several activities that will be executed and place them inside the Compensation Handler container. Configure the properties of each activity.**

Note – You do not have to configure any properties for the compensation handler.

Using a Termination Handler

The termination handler is used to control the termination of a running scope. The termination of a running scope happens if a scope or process enclosing it has faulted.

When to Use

When a fault is thrown inside a scope or process, a fault handler associated with the scope or process should be run, but before that all the running activities inside the faulted scope or process should be terminated. If a faulted scope or process has any enclosed scopes which are still running, they also should be terminated. Terminating a scope means terminating activities inside it and executing the termination handler associated with the scope.

Note that a scope can be terminated only if, it is either running normally, is running its compensation handler or termination handler. A completed scope as well as a scope that is faulted or is running its fault handlers cannot be terminated.

The termination handler is a container for the activities that will be performed in case a scope is terminated. You can add one termination handler for a scope.

If a fault occurs inside the termination handler of a scope, the fault is not propagated to the enclosing scope.

▼ To add a Termination Handler to Scope or Process elements:

- 1 In the Design view right-click the Scope element and choose Add > Termination Handler. An empty container element appears.
- 2 From the Palette, drag one or several activities that will be executed and place them inside the Termination Handler container. Configure the properties of each activity.

Note – You do not have to configure any properties for the termination handler.

Using Correlation

Correlation mechanism is used to route messages to the right processes. The section describes how to define correlation.

[“Understanding Correlation. Using the Correlation Wizard” on page 109](#)

Understanding Correlation. Using the Correlation Wizard

The BPEL Service Engine runtime uses a mechanism called *correlation* to track the multiple, long-running exchanges of messages that typically take place between a BPEL process and its partner services. The correlation mechanism helps to route messages to appropriate process instances.

A message in a conversation is connected with a composite value made up of one or more properties defined in a WSDL file. A property is a field within a message identified by a query. Queries are specified by special constructs called property aliases.

Thus, correlation sets are used to support stateful collaboration between web services in a standardized, implementation independent way. Correlation sets rely on the correlation data tokens stored in the message envelopes, headers, or business documents themselves. The declaration of correlation relies on the declarative properties of messages.

The following terms apply to correlation:

- **Property** — A property is an arbitrarily named token. It must be a simple type. It is defined in a WSDL file.
- **Property alias** — A property alias is a rule that tells the BPEL runtime how to map data from a message into a property value. You can define several property aliases for a property that will be used as a correlation value. You would do this if the same property value needs to be mapped from more than one message, which is typical in correlation. For instance, if two different messages have the same part that you want to extract. Then you need one property and two property aliases - one for each message). Property aliases are defined in a WSDL file.
- **Correlation set** — A correlation set is a compound key made up of one or more property values, actually it is a property set. The BPEL runtime uses this key to ensure that messages are routed to the right process instance for a particular conversation. A correlation set is defined in a BPEL file.
- **Correlations** — Correlations mark the activities, identify the correlation sets by name and indicate which correlation sets occur in the messages being sent or received.

Elements That Use and Express Correlation

Correlation sets can be defined for the Process element. The defined correlation sets are then used by message activities (Invoke, Reply, and Receive), which describe a conversation between a process and a partner service.

Correlation sets on Invoke activities are used to verify that outbound messages contain data that is consistent with the data found within specified correlation set instances.

Correlation set names are also used in the onMessage branches of Pick elements and in the onEvent variant of event handlers.

Defining Correlation Using the Correlation Wizard

There are two ways to define correlation:

- Use the Correlation wizard which will automatically perform all the main steps. This is the most easy and convenient way to define correlation. Usually you do not have to know in details how does correlation work. The Wizard will make it for you.
- Define correlation manually

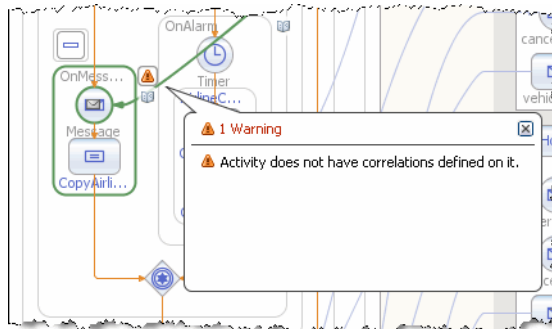
The Correlation Wizard is used to define correlations for two messaging activities, such as Invoke, Reply, Receive, OnEvent or onMessage branch of Pick element.

Note – The wizard is only used to create a correlation. You cannot use the wizard to edit a correlation.

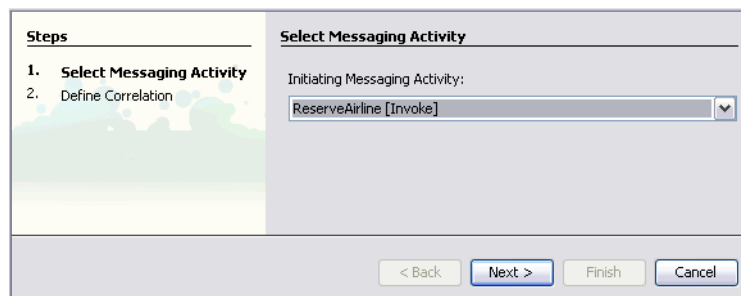
▼ To create correlation using the Correlation Wizard:

- 1 In the Design view, right click the activity that requires correlation and choose Define Correlation.

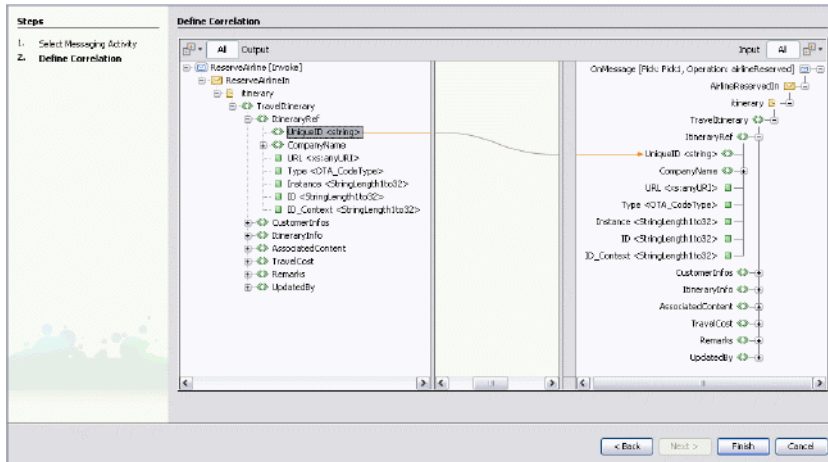
The Correlation Wizard opens. If correlation is required for an activity, the Designer places a warning icon on the diagram.



- 2 In the Correlation wizard, select the messaging activity. From the drop-down list choose an initiating messaging activity. The activity chosen here initiates the correlation set. Click Next.

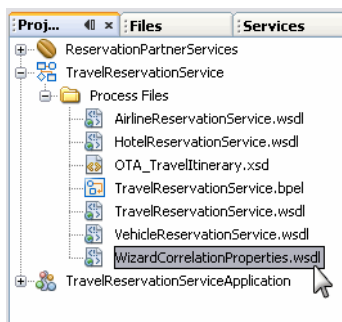


- 3 For Step 2 of the wizard, define the correlation. In the Output pane (left pane) of the wizard the tree structure represents the message that the initiating activity sends or receives. In the Input pane (right pane) The tree structure represents the message passed by the correlating activity. Connect the messages parts that define correlation by selecting the node in the Output (source) pane and dragging your cursor to the appropriate node in the Input (destination) pane.



- 4 Once the correlation is set, the wizard creates properties and property aliases in a WSDL file, defines a correlation set in the BPEL file, and associates the correlation set with the activities you selected.

Note that properties and property aliases are written to a new WSDL file that you can see among the process files of the BPEL Module. The original WSDL file for the partner service is imported to the new WSDL. For all correlation created using the wizard, both properties and property aliases are written to this file. Partner WSDL files are imported. The correlation set defined in the BPEL file refers to the new WSDL.



▼ Defining Correlation Manually

- 1 Define one or more properties in the WSDL file using the WSDL Editor or add a property to a WSDL file.
- 2 Define property aliases in the WSDL file using the WSDL Editor add a property to a WSDL file.
- 3 Define a correlation set for the Process in the BPEL file, using one or more of the previously defined properties.

To define a correlation set:

- a. In the Design view, right-click the Process element and choose Add → Correlation Set.
Alternatively, in the BPEL Logical View of the Navigator window, right-click the Correlation Sets node and choose Add Correlation Set.
- b. In the Add Correlation Set dialog box, specify the name for the correlation set and click Add to add properties.
- c. In the Property Chooser dialog box, expand the WSDL file node, and select a property to add to the set.
- d. (Optional) Clear the Show Imported Files Only checkbox to view the contents of non-imported WSDL and XML schema files.
By default, the Property Chooser dialog box only shows those files that have already been referenced in the process. However, the project may contain other .wsdl and .xsd files which have not yet been imported into the process. If you select a type for the new property that is defined in a non-imported file, the IDE will automatically add the required import to the BPEL process.

The correlation sets defined for the Process have global visibility. The name of a correlation set must be unique among the names of other correlation sets.
- e. Click OK.

- 4 Associate one or more correlation sets with a message that is sent or received in an Invoke, Receive, Reply, or Pick activity.
 - a. In the Design view, double-click an element (Invoke, Receive, Reply, the On Message branch of Pick, or the On Event branch of an Event Handlers container element).
 - b. In the Property Editor, select the Correlations tab and click Add.
 - c. In the Choose a Correlation Set dialog box, expand the Correlation Sets node, select the correlation set and click OK.

- d. **Choose the Initiate attribute for this correlation set from the Initiate drop-down list. You can select one of the following options:**
 - **Yes.** The activity must attempt to initiate the correlation set.
 - **Join.** The activity must attempt to initiate the correlation set if the correlation set is not yet initiated.
 - **No.** The activity must not attempt to initiate the correlation set. This is the default option.
- e. **For an Invoke activity, specify the message pattern.**

From the Pattern drop-down list, select a pattern attribute to indicate whether the correlation applies to the outbound message (request), inbound message (response), or both (request-response).
- f. **(Optional) Add more correlation sets as needed and click OK.**

Validation

The BPEL Designer has a built-in BPEL code validation functionality that helps developers create well-formed, valid and standard-compliant code. The code is checked for errors and the user is notified if validation fails.

Validation Criteria

The Validator checks the BPEL process in accordance with the following criteria:

1. For conformance with the BPEL 2.0 schema.

See the “[Troubleshooting](#)” on page 170 section of this guide for more information about using BPEL schemas different from the BPEL 2.0 specification.
2. For compliance with static analysis rules defined in the WS-BPEL 2.0 specification.
3. For broken references.
4. For constructs that are valid per the BPEL 2.0 specification but are not yet supported by the Sun BPEL Service Engine.

Validation Types

The BPEL Designer provides two types of validation:

Real-time validation

This type of validation is invoked automatically and does not require any explicit actions from the user. Only the current file is checked. The validation is performed in accordance with all the criteria mentioned above, except for validation for conformance with the BPEL 2.0 schema.

Explicit validation

This type of validation requires that the user explicitly invoke the validation process. All imported XSD and WSDL files are also checked. The validation is performed in accordance with all the criteria mentioned above.

To invoke explicit validation, do one of the following:

- In the Source view, right-click the source to invoke the pop-up menu and choose Validate XML (Alt-Shift-F9)
- In the Design view, click the Validate XML button (Alt-Shift-F9) on the Editor toolbar.



Notifications

The user is notified about validation errors or success in the Output window, in the Design view, and in the Navigator.

The Output window

The results of validation appear in the Output window if validation has been invoked explicitly. If validation fails, the Output window contains errors and/or warnings:

```

Output - XML check
home/dm203010/TravelReservationService/TravelReservationService/src/AirlineReservatio
Model source provides no javax.xml.transform.Source in lookup, relative resource res
home/dm203010/TravelReservationService/TravelReservationService/src/VehicleReservatio
Model source provides no javax.xml.transform.Source in lookup, relative resource res
home/dm203010/TravelReservationService/TravelReservationService/src/HotelReservatio
Model source provides no javax.xml.transform.Source in lookup, relative resource res

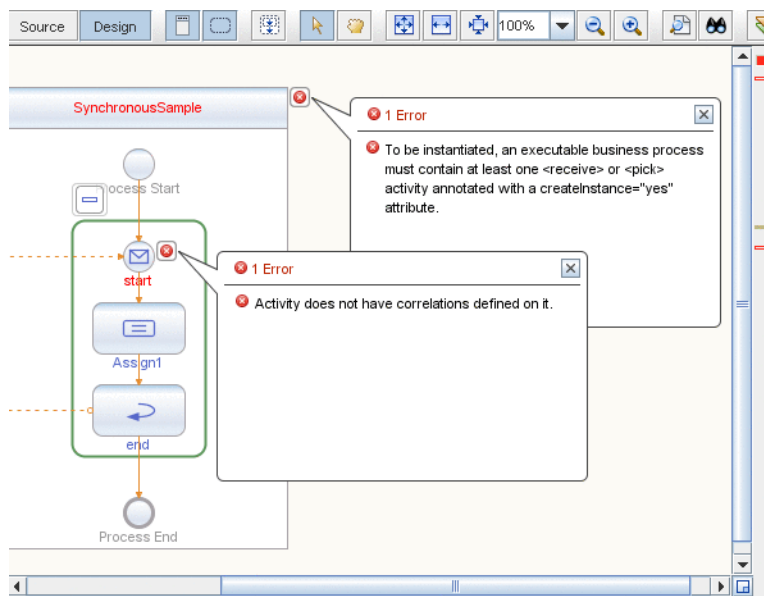
1 Error(s), 5 Warning(s).
XML validation finished.
  
```

If validation is successful, there are no warnings or errors in the Output window.

The Design view

The Design view shows the results of both real-time and explicit validation in callout windows on the diagram and the error stripe.

In the diagram, a red cross next to an element on the diagram means that the element has not passed validation and the output contains errors. A yellow triangle with an exclamation mark means that the element has not passed validation and the output contains warnings. If there are both errors and warnings, the Design view shows a red cross. If you click the cross or the triangle, a callout window appears with a list of errors and/or warnings:

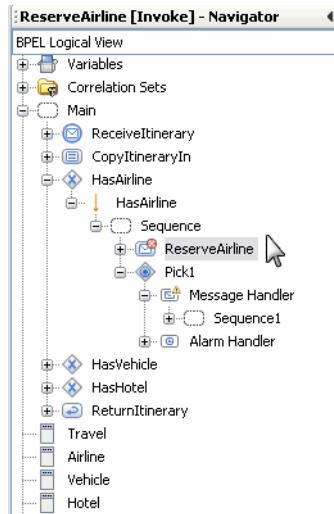


The callout window includes messages related to validation in accordance with all the criteria listed above. Messages related to the real-time validation are constantly updated.

In the Design view, validation results are also shown by the error stripe, which is a strip to the right of the scroll bar that contains red marks if some elements have not passed validation. The error stripe represents the entire diagram, not just the portion that is currently displayed. You can immediately see if your BPEL process contains errors without having to scroll through the entire diagram. You can click a red mark to jump to the element that causes problems. If no errors are detected, the small square in the error stripe is green.

The Navigator window

The Navigator window shows the results of both real-time and explicit validation by adding a red cross or a yellow triangle to the element's icon if validation has failed. For example, in the screenshot below, the `AirlineReserved` receive activity has not passed validation and the output contains errors.



BPEL Process Logging and Alerting

The Sun BPEL Service Engine provides you with the ability to trace the message or expression values during the process execution. The Logging and Alerting feature make use of standard WS-BPEL extension mechanism. Logging and alerting are supported for almost all BPEL activities.

The NetBeans IDE provides the ability to define logging and alerting for the process activities.

- Logging is used to write specified expression values or partner links endpoint reference information to the server log.
- Alerting allows you to receive an alert with this information.

After you set the logging or alerting conditions and the BPEL process is executed, specified expression values are written to the server log file or an alert is sent to the user, depending on the log level.

Both logging and alerting are defined in the Logging mapper. The Logging mapper is available from the Design or Source view menu bar.

Defining Logging

When defining logging for an activity you can trace the value of the following components :

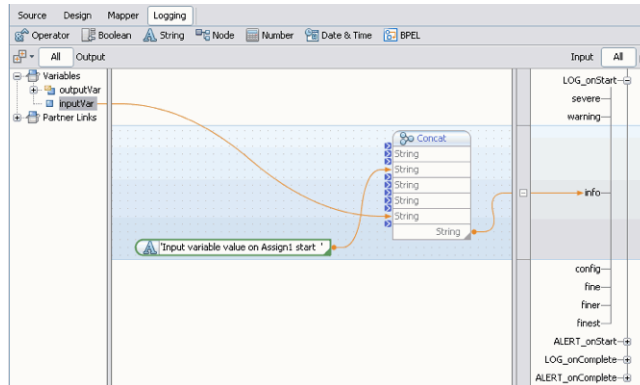
- Variable
- Part
- Expression

In the mappings you can use one or more XPath functions from the menu bar.

▼ To log the variable value:

- 1 **On the diagram, select an activity. The logging will be performed in connection with the activity execution.**
- 2 **Go to the Logging tab of the BPEL Editor. The Logging mapper opens. You can also open the Logging mapper by right clicking the activity and choosing Go To → Logging (Alt-L).**
- 3 **In the source tree pane, expand the variables tree until the variable to be traced is visible.**
- 4 **In the destination tree pane expand the activity node. The nodes designating the moment of logging become visible.**
- 5 **Choose when the logging entry should be made and expand the appropriate node:**
 - **LOG_onStart.** The variable value is written to the log when the activity starts.
 - **LOG_onComplete.** The variable value is written to the log when the activity execution is complete.
- 6 **Define the level of logging. Drag the connection from the variable to be traced to the appropriate node in the destination tree pane. The following levels of logging are available:**
 - **Severe**
 - **Warning**
 - **Info**
 - **Config**
 - **Fine**
 - **Finer**

To make a search of the value recorded to the log file, you can concatenate the value with the string literal as shown on the figure below.



In the Design view a small icon appears to the lower-right of the activity when it has logging defined. By clicking the icon you can switch to the Logging mapper.



The entry to the log is only made if the log level defined for the variable corresponds to the log level specified for the BPEL Service Engine on the application server.

▼ To set the log level for the BPEL Service Engine

The log level for the BPEL Service Engine is specified in the GlassFish Admin Console. To set the log level for the BPEL Service Engine:

- 1 In the NetBeans Services window, expand the Servers node and ensure that the GlassFish application server is running. A green arrow badge next to the server node indicates that the server is running. If the server is not running, right click the server name and choose Start from the context menu.
- 2 Open the Admin Console in your browser. To do this, follow the steps:
 - Right click GlassFish V2 application server node, and choose Properties from the context menu. The Servers window opens. On the Servers pane, GlassFish V2 should be selected.

- On the Connection tab, copy the contents of the Location field (by default it is localhost:4848).
 - Paste the string to the browser and press Enter. The GlassFish Admin Console opens in the browser window.
- 3 **Log in to the Admin Console using your username and password. By default, the username is admin and the password is adminadmin.**
 - 4 **On the left pane under the JBI node choose Components → sun-bpel-engine. The BPEL service engine properties page opens.**
 - 5 **On the BPEL service engine properties page, select the Loggers tab. On the Loggers tab you can specify log levels for the individual loggers.**
 - 6 **Choose the appropriate log level for the sun-bpel-engine from the drop down list.**

If logging is defined for a process activity, and the log level specified for it corresponds to the log level set for the BPEL SE, after you perform a test run of the process, the selected variable value will be written to the server log file.

Note – The project should be deployed to the application server.

▼ **To view the log file:**

- 1 **In the Services window, under the Servers node, right click GlassFish V2 application server node and choose View Server log from the context menu. The GlassFish server log opens in the Output window. The activity message value will be included in the log, you can use Search to find it. Note, that some overhead information is hidden.**
- 2 **You can also open the log in a text editor and see the full information. Navigate to <application server installation directory>/domains/domain1/log/ and open the server.log file with the text editor. The information provided in the log includes the following points, divided with the vertical bar:**
 - Date and time of the entry
 - Log level
 - Manager type (for logging this is Trace Manager)
 - Thread
 - The message value

Here is the sample of the log entry :


```
[#|2008-03-25T09:26:18.796+0300|INFO|sun-appserver9.1|com.sun.jbi.engine.bpel.core
version="1.0" encoding="UTF-8"?><jbi:message
xmlns:msgns="http://localhost/SynchronousSample/SynchronousSample"
name="input1" type="msgns:requestMessage" version="1.0"
xmlns:jbi="http://java.sun.com/xml/ns/jbi/wsd1-11-wrapper"><jbi:part><syn:typeA
xmlns:syn="http://xml.netbeans.org/schema/SynchronousSample">
<syn:paramA>Hello World</syn:paramA>
</syn:typeA></jbi:part></jbi:message>|#]
```

Defining Alerting

Alerting feature enables you to get notification in case specified events happen. Alerting events are connected with the execution of the process activities.

As a general rule-of-thumb, whenever you log you may also want to consider sending an alert notification of the appropriate severity as you see fit. Fatal, Critical, and Major Alert Notification Severities map well to a SEVERE logging category. The Minor and Warning Alert Notification Severities map well to a WARNING logging category. The Info Alert Notification Severity maps well to an INFO logging category.

The general workflow for defining alerting is as follows:

1. Set the alert level for your activity. This is done from the Logging Mapper, similar to the way you define Logging. Map the variable in the Output pane for which you want an alert, to the appropriate Alert activity and alert level, for example, ALERT_onComplete - major.

The Alert notification levels are:

- Fatal
 - Critical
 - Major
 - Minor
 - Warning
2. Ensure the application server is running and deploy the project.
 3. From the Admin Console, choose or create MBean client and subscribe to getting event notifications. The client will extract alerting messages and perform specified actions (write to log/send e-mail/do nothing).
 4. Run the process and get notified.

Configuring the BPEL Service Engine Runtime Properties

The BPEL Service Engine runtime properties can be configured from the NetBeans IDE, or from a command prompt (command line interface) during installation.

Configuration Properties are Grouped by Purpose

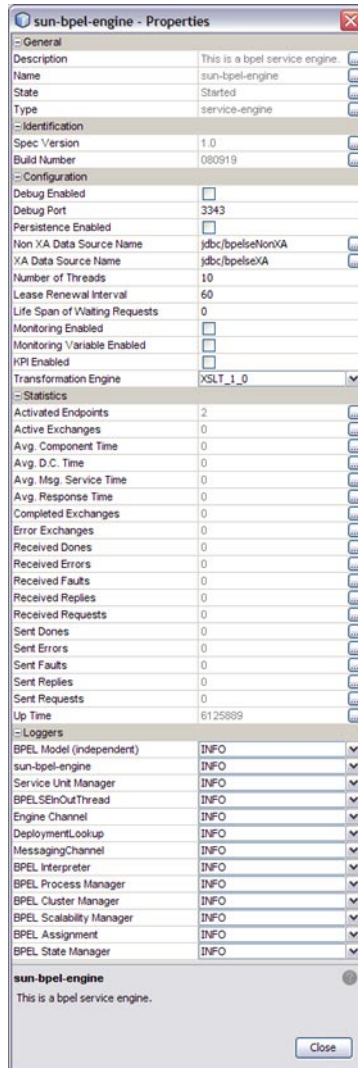
The configuration properties are grouped together for specific purposes.

- Debugging the BPEL Service Engine business process:
 - Debug Enabled
 - Debug Port
- Persistence and Recovery configuration:
 - Persistence Enabled
 - Non XA Data Source Name
 - XA Data Source Name
- Tuning:
 - Number of Threads
 - Lease Renewal Interval
 - Life Span of Waiting Requests
- Monitoring:
 - Monitoring Enabled
 - Monitoring Variable Enabled
 - KPI Enabled

Accessing the BPEL Service Engine Runtime Properties

To display or edit the properties in the NetBeans IDE, do the following:

1. From the Services tab of the NetBeans IDE, expand the Servers node.
2. Start your application server, for example GlassFish v2. To do this, right-click your application server and select Start from the shortcut menu.
3. Under the application server, expand the JBI → Service Engines nodes and select the BPEL Service Engine. The current BPEL Service Engine properties are displayed at the right side of the NetBeans IDE. You can also double-click the BPEL Service Engine to open a properties window.
4. Edit the properties as needed. To apply any changes you make to the runtime BPEL Service Engine properties, stop and restart the BPEL Service Engine.



Runtime Property Descriptions

The following table include descriptions for the BPEL Service Engine runtime properties

Property Name	Description	Default Value
General Properties		

Property Name	Description	Default Value
Description	Description of the JBI Component	This is a bpel service engine.
Name	Name of the JBI Component. Specifies a unique name in the JBI environment. If you are installing more than one BPEL Service Engine in a JBI environment, make sure that each is unique. This can be changed in the descriptor (jbi.xml) for the component. When the service unit deploys the component, it is matched with target component name defined in its descriptor – jbi.xml	sun-bpel-engine
State	State of the JBI Component. Started, Stopped, or Shutdown	Started
Type	Type of the JBI Component (service-engine or binding-component)	service-engine
Identification Properties		
Build Number	Date and time stamp for the current build	<build_number>
Spec Version	BPEL specification fully supported by this build	<spec_version>
Configuration		
Debug Enabled	Specifies whether the debugger can be used. A Selected the checkbox indicates that you can attach the debugger and debug the business process definition	Select the checkbox to enable. The default is unchecked
Debug Port	Specifies the port number at which the BPEL Service Engine listens for the debugger UI to connect. The default value is sufficient in most cases. If more than on instance of the BPEL Service Engine is running on the same computer, make sure that unique, non-conflicting ports are assigned to each. Do not allow other applications to use these assigned ports	3343
Persistence Enabled	When the checkbox is checked, persistence is enabled. The BPEL Service Engine persists the state of the business process instance at the configured data source for recovery in the event of a crash	Select the checkbox to enable. The default is unchecked

Property Name	Description	Default Value
Non XA Data Source Name	Specifies the non-XA data source where the BPEL Service Engine persists the state of the business process, to be used for recovery in the event of a crash	Example: jdbc/bpelseNonXA
XA Data Source Name	Specifies the XA data source where the BPEL Service Engine persists the state of the business process, to be used for recovery in the event of a crash	Example: jdbc/bpelseXA
Number of Threads	Specifies the number of threads allowed to execute BPEL definitions. Configure this to equal the number of CPUs for the system to achieve maximum throughput	10
Lease Renewal Interval	This property is only used for clustered environments. It specifies the interval (in seconds) at which the BPEL Service Engine renews its lease to continue to own the business process instance it is currently executing	60 (seconds)
Life Span of Waiting Request	Specifies the life span of a request after it is received. If the request is not consumed within the specified time, the BPEL Service Engine responds with an error for that request. The time is specified in seconds. A value of "0" indicates that the life span is indefinite	0
Monitoring Enabled	Specifies whether the Service Engine business process instance and activities at the configured data source are monitored	Select the checkbox to enable. The default is unchecked
Monitoring Variable Enabled	Specifies whether the Service Engine enables changes to the monitoring variable value at the configured data source. This property is only applicable when Monitoring Enabled is checked	Select the checkbox to enable. The default is unchecked
KPI Enabled	Specifies whether the Service Engine posts real-time KPI (key performance indicator) events	Select the checkbox to enable. The default is unchecked
Transformation Engine	Specifies which transformation processor is used to execute XSL stylesheets. The choices are XSLT_1_0 and XSLT_2_0	XSLT_1_0

Property Name	Description	Default Value
Statistics		
Activated Endpoints	The number of activated endpoints	0
Active Exchanges	The number of active exchanges	0
Avg. Component Time	The average message exchange component time in milliseconds	0
Avg. D.C. Time	The average message exchange delivery channel time in milliseconds	0
Avg. Msg. Service Time	The average message exchange message service time in milliseconds	0
Avg. Response Time	The average message exchange response time in milliseconds	0
Completed Exchanges	The total number of completed exchanges	0
Error Exchanges	The total number of error exchanges	0
Received Dones	The total number of received dones	0
Received Errors	The total number of received errors	0
Received Faults	The total number of received faults	0
Received Replies	The total number of received replies	0
Received Requests	The total number of received requests	0
Sent Dones	The total number of sent dones	0
Sent Errors	The total number of sent errors	0
Sent Faults	The total number of sent faults	0
Sent Replies	The total number of sent replies	0
Sent Requests	The total number of sent requests	0
Up Time	The up time of this component in milliseconds	0

Property Name	Description	Default Value
Loggers	Specifies the user-designated level of logging for each event. Each logger can be set to record information at any of the following levels: <ul style="list-style-type: none"> ■ FINEST: provides highly detailed tracing ■ FINER: provides more detailed tracing ■ FINE: provides basic tracing ■ CONFIG: provides static configuration messages ■ INFO: provides informative messages ■ WARNING: messages indicate a warning ■ SEVERE: messages indicate a severe failure ■ OFF: no logging messages 	
BPEL Model (independent)	com.sun.bpel.model.impl	INFO
sun-bpel-engine	com.sun.jbi.engine.bpel	INFO
Service Unit Manager	com.sun.jbi.engine.bpel.BPELSEDeployer	INFO
BPELSEInOutThread	com.sun.jbi.engine.bpel.BPELSEInOutThread	INFO
EngineChannel	com.sun.jbi.engine.bpel.EngineChannel	INFO
DeploymentLookup	com.sun.jbi.engine.bpel.com.sun.jbi.common.descriptor.DeploymentLookup	INFO
MessagingChannel	com.sun.jbi.engine.bpel.com.sun.jbi.common.messaging.MessagingChannel	INFO
BPELInterpreter	com.sun.jbi.engine.bpel.core.bpel.engine.impl.BPELInterpreter	INFO
BPEL Process Manager	com.sun.jbi.engine.bpel.core.bpel.engine.impl.BPELProcessManagerImpl	INFO
BPEL Cluster Manager	com.sun.jbi.engine.bpel.core.bpel.engine.impl.BPELClusterManager	INFO
BPEL Scalability Manager	com.sun.jbi.engine.bpel.core.bpel.engine.impl.BPELScalabilityManager	INFO
BPEL Assignment	com.sun.jbi.engine.bpel.core.bpel.model.runtime.impl.AssignUnitImpl	INFO
BPEL State Manager	com.sun.jbi.engine.bpel.core.bpel.persist.impl.StateManagerImpl	INFO

BPEL Service Engine Deployment Artifacts

The BPEL Service Engine requires the following artifacts to execute a business process:

- BPEL Documents: define the activity sequence to execute.
- WSDL Documents: define the contract between the business process and partner services.
- Optional XSDs: define exchanged XML documents; XSDs can be defined inline within WSDL documents.

These artifacts are packaged into a service unit, and in turn packaged into the service assembly, along with other JBI component's service units, based on the internal/external partner services requirements.

Service assemblies and service units can be deployed to JBI runtime and corresponding components using Ant scripts provided with the NetBeans IDE. The JBI `DeploymentServiceMBean` interprets the deployment descriptor, `jbi.xml`, and deploys the service unit to the associated component.

Testing and Debugging BPEL Processes

The section describes how to test and debug BPEL processes.

[“Testing a BPEL Process” on page 128](#)

[“Debugging BPEL Processes” on page 131](#)

[“BPEL Debugger Windows” on page 139](#)

Testing a BPEL Process

Testing a deployed business process application involves using test cases that act as remote partner services. These test cases send SOAP messages to the BPEL Service Engine.

In simple words, the interaction process is as follows:

- The BPEL Service Engine receives the SOAP message, creates an instance of the BPEL process, and starts executing the instance. A BPEL process can have many running instances.
- The BPEL Service Engine receives a message and, using correlation, routes it to the appropriate instance of the process. If an instance does not yet exist, a new instance is created.

To test-run a deployed business process application, you need to configure test cases to act as remote partner services sending SOAP messages to the BPEL Service Engine.

Creating and Running a Test Case

In order to obtain test results you must do the following:

1. [Add a test case to your BPEL project.](#)
2. [Set the test properties.](#)
3. [Customize the test input file.](#)
4. [Run the test case.](#)

All steps in this section assume the following:

- You have already created a new BPEL Module Project containing a WSDL file that defines an operation you want to test.
- You have successfully completed a build of the BPEL Module Project.
- You have added your BPEL Module project to a Composite Application project as a JBI Module.

Adding a Test Case to your BPEL Project

▼ To add a test case and bind it to a BPEL operation

- 1 In the NetBeans IDE Projects window, expand your Composite Application project to expose the Test folder.
- 2 Right-click Test, and choose New Test Case from the pop-up menu. This launches the New Test Case wizard.

This launches the New Test Case wizard.

- 3 In the Enter the Test Case Name step, enter a name for the test case and click Next.
- 4 In the Select the WSDL Document step, open the BPEL Module project, select the WSDL file containing the operation you want to test, and click Next.
- 5 In the Select the Operation to Test step, select the operation you want to test, and click Finish.
In the Projects tree, a new folder is created under the Test node, containing two files: Input .xml and Output .xml.

Note – If you view the test case in the Files window, you see Concurrent.properties as a third file.

Setting the Test Properties

▼ To set the test properties

- 1 In the Projects window, under the Composite Application > Test node, right-click the node for the test case and choose Properties from the pop-up menu.
- 2 Set the properties of the test case as follows:
 - **Description** — string
 - **Destination** — URL (from the .wsdl file's <soap:address location="THIS"> tag)

Identifies the location of the web service to be tested.

- **SoapAction** — (default: blank)
- **Input File** — (read-only; generated by system)
Name of the input file. This file contains the input data for the test case.
- **Output File** — (read-only; generated by system)
Name of the output file. This file contains the output data for the test case.
- **Concurrent Threads** — integer; default = 1
Each thread can invoke the test case multiple times (see the following property). Thus, if conc=2 and inv=3, the test case will be run 6 times (two threads, each run thrice).
- **Invokes Per Thread** — integer; default = 1
Number of times each thread invokes the test case.
- **Test Timeout (sec)** — integer; default = 30
How long each thread has to finish. If it does not finish in the allotted time, then an exception is thrown.
- **Calculate Throughput** — boolean
- **Comparison Type** — drop-down list with the following options:
 - **identical** — Considers the output and actual output as a stream of characters.
 - **binary** — Considers the output and actual output as a stream of bytes.
 - **equals** — Considers the output and actual output as a XML documents.
- **Feature Status** — drop-down list with the following options:
 - **progress** — Marks test completion as "success", regardless of actual outcome.
 - **done** — Records actual outcome of test.

Customizing Test Input

▼ To customize test input

- 1 In the Projects window, expand the Test node and the node for a specific test case.
- 2 Right-click Input and click Edit.
- 3 Modify the contents as needed. For example, wherever you see `<value?>string?</value>` click within `?string?` and replace it with a string of any length. However, within such strings, do not include the characters "`<`" (less-than sign) or "`&`" (ampersand) unless you use them with XML semantics.
- 4 When you are satisfied, click Save.

5 Right-click `Output.xml` and choose **Edit to examine the contents:**

- If it is empty, this is a special state that triggers a special operation when the test is run.
- Each time the test is run, the current output is compared to the contents of `Output.xml`. If any differences are detected, they will be stored in the `Actual_ymmddhhmmss.xml` file under the test case folder. However, in the special case where `Output.xml` starts null, then the output is written to `Output.xml`.
- In each run after the first, assuming `Output.xml` is no longer null, its contents are preserved. In other words, a previous output is never overwritten by new results.

Running Test Cases

To run a single test case

To run a single test case, right-click the node for your specific test case, and choose **Run**. Check the **Output** window for the results.

To run all test cases in a project

To run all test cases in a project, right-click your **Composite Application** project and choose **Test (Alt+F6)** from the pop-up menu. Check the **Output** window for the results.

Looking at Test Case Results

The first time you run your test, the results correctly report that it has failed. This happens because the output produced does not match the (empty) `Output.xml` file, and the file's null content is replaced with the output of the first run. If you run the test again without changing the input, the second and subsequent runs report success, since the output matches the contents of `Output.xml`.

Test results are displayed in the **Output** window. Detailed results are also displayed in the **JUnit Test Results** window, which opens automatically when you run a test case. If you change the value in the `Input.xml` and re-run the test:

- If the `feature-status` property is set to `progress`, then the test indicates success even though a mismatch occurred.
- If the `feature-status` property is set to `done`, then the test indicates failure.

If you right-click the test case node and click **Diff** in the pop-up menu, the window displays the difference between the latest output and the contents of `Output.xml`.

Debugging BPEL Processes

Debugging a BPEL process follows the same general principles as debugging a Java application. Debugging a BPEL process involves setting breakpoints in the source code and executing the

process step-by-step within a debugging session. The BPEL Debugger provides a visual representation of the BPEL process execution. This enables you to view and change variables, monitor the results of executing expressions, and use fault breakpoints to monitor the state of variables before a fault is thrown.

Steps in Debugging BPEL Processes

The main steps in debugging BPEL processes are:

1. Confirm that the GlassFish application server has started.
2. Create test cases.
For sample processes, test cases are automatically created; for new projects, you need to create at least one test case.
3. Open the BPEL process file either in the Source view or Design view.
4. Set breakpoints in the code or on the diagram. Optionally, add watches for XPath expressions in your process or add fault breakpoints.
5. Start a debugging session. Watch the BPEL Debugger Console window for confirmation that the debugging session has started.
6. Within the debugging session, run one or several test cases.
7. View execution of BPEL processes on the diagram in the Design view or in the BPEL Process Execution window and view running instances of BPEL processes in the BPEL Process Instances window.
8. When an instance stops at a breakpoint, step through the code or the diagram, examine the values of variables in the BPEL Variables window, or observe the values of XPath expressions in the Watches window.
9. Finish the debugging session.

Starting and Finishing a BPEL Debugging Session

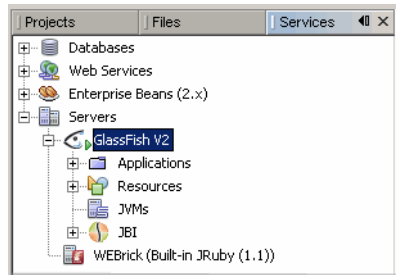
A debugging session begins when you connect the BPEL Debugger to the BPEL Service Engine. Only one debugging session can be running with the BPEL Service Engine at a given time.

After a BPEL debugging session starts, you can execute process instances step-by-step, inspecting the values of BPEL variables and XPath expressions in the Local Variables and Watches windows. You can monitor the execution of a BPEL process within a debugger session on the diagram in the Design view: the activities that are being executed are highlighted on the diagram as the current execution position advances. The BPEL Process Execution window also shows the execution of the BPEL process.

▼ To prepare the debugging environment

- 1 In the Services window, make sure that the GlassFish V2 Application Server is running. The Application Server is running if it has subnodes and is marked with a green triangle.

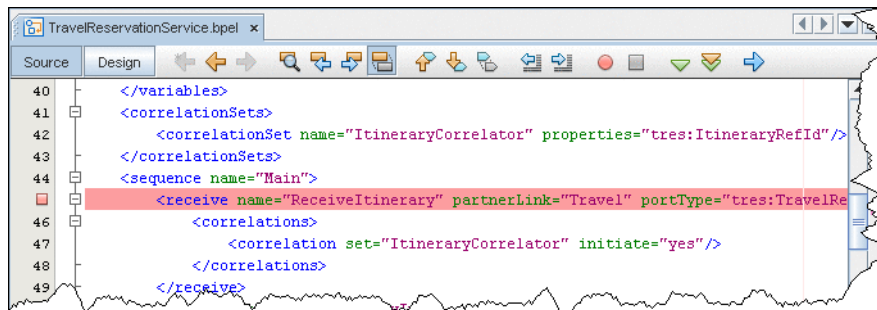
If the server is not started, right-click it and choose Start from the pop-up menu.



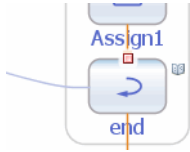
- 2 In the IDE, open the BPEL process in either the Source or Design view.

- 3 Set breakpoints in the BPEL process.

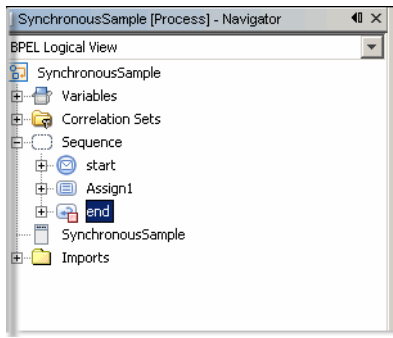
To set breakpoints in the Source view, click next to the line where you want to set the breakpoint.



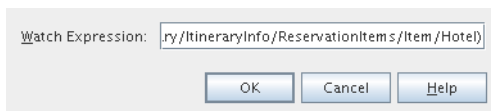
To set breakpoints on the diagram, switch to the Design view, right-click the element and choose Toggle Breakpoint from the pop-up menu. A red square appears at the top of the element with a breakpoint.



The Toggle Breakpoint pop-up menu command is also available for the elements in the Navigator BPEL Logical View. For the elements with breakpoints, the Navigator shows a small red box (ReceiveItinerary).



- 4 **Optionally, you can add watches to monitor XPath expressions. To add a watch, copy the XPath expression you want to monitor, choose Run → Add Watch from the main menu, and paste the expression into the Watch Expression field. Click OK.**



Note – You can also add XPath expressions that are not present in the code, but that would be valuable from the debugging point of view.

▼ To start and finish a debugging session on the BPEL Engine

- 1 **In the Projects window, right-click the Composite Applications project you want to debug and choose Debug (BPEL) from the pop-up menu.**

A debug session is established on the BPEL Service Engine.

11:35:17 Connecting to localhost:3343

- Enables debugging with the BPEL Service Engine (sets the DebugEnabled property of the BPEL Service Engine to true)
- Builds the Composite Application project and all JBI Modules added to this project
- Deploys the Composite Application project to the BPEL Service Engine
- Starts the debugging session by connecting the BPEL Debugger to the BPEL Service Engine

Therefore, whenever you start a debugging session you can be sure that the latest version of the BPEL process is deployed on the BPEL Service Engine.

Now you can run a test case and monitor the execution of the BPEL process until it stops or reaches a breakpoint. As the process advances, the current context is displayed on the diagram and in the BPEL Process Execution window.

If you have several debugging sessions (you may have a Java debugging session running at the same time) and want to change the current session, double-click the name of this session in the Sessions window. Alternatively, right-click the session you want to make current and select Make Current. This session becomes current and the BPEL Process Instances, Watches and Local Variables Windows are updated to show the data related to the new current session.

When you want to finish a debugging session, open the pop-up menu for the session you want to stop and choose Finish in the Sessions window or select Finish Debugger Session on the toolbar. A message that the debugging session is finished is displayed in the BPEL Debugger Console.

To finish all debugging sessions, in the Sessions window, right-click any session and choose Finish All.

2 Watch the BPEL Debugger Console window for confirmation. The connection might take some time to complete. When it is successfully completed, you can see the new session in the Sessions window and the following messages in the BPEL Debugger Console:

- 11:35:17 Connecting to localhost:3343
- 11:36:19 Debug session started

The Debug (BPEL) command performs the following actions:

- Enables debugging with the BPEL Service Engine (sets the DebugEnabled property of the BPEL Service Engine to true)
- Builds the Composite Application project and all JBI Modules added to this project
- Deploys the Composite Application project to the BPEL Service Engine
- Starts the debugging session by connecting the BPEL Debugger to the BPEL Service Engine Therefore, whenever you start a debugging session you can be sure that the latest version of the BPEL process is deployed on the BPEL Service Engine

3 Now, run a test case and monitor the execution of the BPEL process until it stops or reaches a breakpoint.

As the process advances, the current context is displayed on the diagram and in the BPEL Process Execution window.

If you have several debugging sessions (you may have a Java debugging session running at the same time) and want to change the current session, double-click the name of this session in the Sessions window. Alternatively, right-click the session you want to make current and select Make Current. This session becomes current and the BPEL Process Instances, Watches and Local Variables Windows are updated to show the data related to the new current session.

4 To finish a debugging session, open the pop-up menu for the session you want to stop and choose Finish in the Sessions window, or select Finish Debugger Session on the toolbar.

A message that the debugging session is finished is displayed in the BPEL Debugger Console.

5 To finish all debugging sessions, in the Sessions window, right-click any session and choose Finish All.

Using Breakpoints to Debug BPEL Processes

Breakpoints are used to instruct the BPEL Debugger to stop execution at the specified place of a BPEL process. When a BPEL process instance reaches a breakpoint, it becomes suspended and you can step into the code, change the current process instance in the BPEL Process Instances window, track the execution of the process instance in the BPEL Process Execution window and in the Design view, examine the values of variables in the Local Variables window, view the process partner links in the Partner Links window and view the values of XPath expressions in the Watches window.

You can also use fault breakpoints to check the values of variables before a fault is thrown.

To view and organize the breakpoints currently set in the IDE, open the Breakpoints window by choosing Windows → Debugging → Breakpoints (Alt-Shift-5). For each breakpoint, you can see the name of the file and the line where this breakpoint is located. In the Breakpoints window you can enable and disable breakpoints by checking or removing the checkbox in the Enabled column.

▼ To set a breakpoint in a BPEL process

1 In the IDE, open the BPEL file in either the Source or Design view.

2 Do one of the following:

- In the Source view, click the left margin of the row where you want to place a breakpoint.
- In the Design view, right-click an element where you want to place a breakpoint and choose Toggle Breakpoint (Ctrl-F8).

In the Design view, breakpoints are displayed as small red squares on top of specific elements. In the Source view, breakpoints are shown as red squares on the left margins of code lines.

Alternatively, you can set and remove breakpoints in the BPEL Logical view of the Navigator window by choosing Toggle Breakpoint from the pop-up menu. In the Navigator window breakpoints are shown as small red squares attached to elements.

Once the project has reached the breakpoint it is suspended. You can manage the subsequent execution using the commands available in the Run menu or as buttons on the toolbar.

Debugging Commands

The following commands are available from within the debugging session:

- **Pause.** Once the user activates this action, the process will continue till it reaches the first element on which it can stop. If there is no current process instance, the debugger will wait for the first execution event from any process instance.
- **Continue (F5).** Once the process has reached the breakpoint or is paused you can choose Continue. This action causes the current process instance to run until it encounters the next breakpoint or until the instance completes. The state of the instance changes to Running.
- **Step Into (F7).** Steps to the next BPEL activity. As you step, the current line indicator advances, the current position is highlighted on the diagram, and the contents of the BPEL Debugger windows change accordingly. If the current activity has any enclosed elements, the process will step to the first enclosed element. Sometimes it is not visible on the diagram but is reflected in the BPEL Process Execution window. For example, if an Assign activity has <copy> element inside it, from the Assign the process will step to Copy.
- **Step Over (F8).** Steps to the next BPEL activity of the same level as the current activity. If the current activity has any enclosed elements, they all are executed without suspension.
- **Step Out (Ctrl-F7).** Steps to the next higher level activity of the process. For example, an Assign activity has several Copy elements inside. If one of the Copy elements is the current activity, performing Step Out will move you to the next activity of the Assign level and you will not have to step through all the Copy elements.
- **Run to Cursor (F4).** Runs the BPEL process to the position selected in the Navigator window (BPEL Logical View), on the diagram (in the Design view) or to the cursor location in the Source view. When the location of the cursor is reached, the process instance becomes suspended.

▼ To remove a breakpoint from the BPEL process

- 1 In the Source view, On the diagram, click the left margin of the line that contains the breakpoint.

- 2 In the Breakpoints window, right-click the breakpoint you want to remove and choose Delete. Choosing Delete All from the pop-up menu removes all breakpoints currently set in the NetBeans IDE.
- 3 3. In the Design view, right-click the element that has a red breakpoint mark and choose Toggle Breakpoint.

To disable a breakpoint

To disable a breakpoint do one of the following:

- On the diagram, click on the small red square indicating the breakpoint. This disables the breakpoint but does not remove it completely.
- In the Breakpoints window, clear the Enabled checkbox for the breakpoint you want to disable.

Group operations over breakpoints

The toolbar contains three buttons for group operations over the process breakpoints.

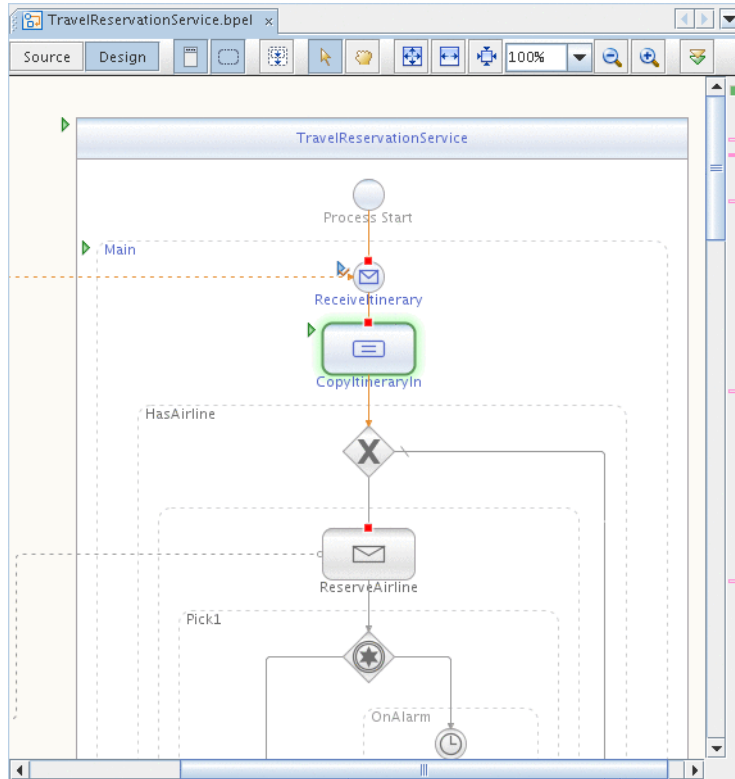
- **Enable Breakpoints for Selected Element.** Enables all the breakpoints for the selected element and its enclosed elements.
- **Disable Breakpoints for Selected Element.** Disables all the breakpoints for the selected element and its enclosed elements.
- **Delete Breakpoints for Selected Element.** Deletes all the breakpoints for the selected element and its enclosed elements. If no element is selected on the diagram, the Process element is considered selected for the operations.

Monitoring Execution of BPEL Processes

When a running process reaches a breakpoint, the Design view highlights the current position of the debugger and uses colors to differentiate between the states of BPEL activities. As the process advances, the colors and icons for the activities on the diagram are updated to reflect the execution progress.

On the diagram, the following notation is used:

- Green color (glowing). The breakpoint set for the activity is reached.
- Gray color (grayed-out effect). The activity has not been executed yet.
- Green triangle. The activity is now being executed.
- Blue triangle. The activity has been successfully completed.



You can also monitor the execution of current BPEL process instances in the BPEL Process Execution window (see below).

BPEL Debugger Windows

When a debugging session starts, debugger windows are displayed below the editing area. The Sessions, BPEL Process Instances, BPEL Variables, and BPEL Process Execution windows contain information related to BPEL processes running within the current debugging section.

If a debugger window is not displayed, choose **Window > Debugging > *window-name*** (for example, **Window > Debugging > BPEL Process Instances**).

The Breakpoints and Watches are standard IDE debugger windows. They display all breakpoints and watches set in the IDE.

Sessions Window

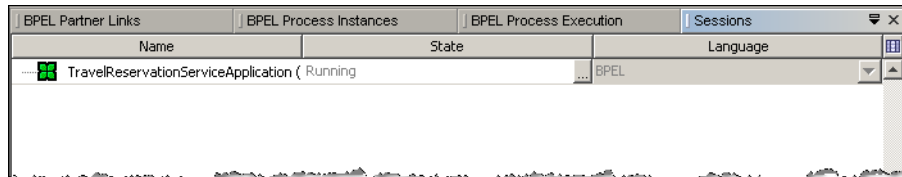
The Sessions window lists all open debugging sessions, including Java and BPEL debugging sessions. For the BPEL Service Engine, only one session can be started. However, the Sessions window also displays other open debugging sessions, such as Java sessions. Only one of the open debugging sessions can be current, and it is shown in bold. Other debugger windows, such as BPEL Process Instances, BPEL Process Execution, and BPEL Variables, display data related only to the current debugging session.

The information provided for each session includes:

- **Name.** The name of the session.
- **State.** The current state of the session. Sessions can be starting or running.
- **Language.** The language of the application debugged in this session.

You can perform the following actions on sessions available in the pop-up menu:

- **Make current.** Makes the selected session current.
- **Finish.** Finishes the selected session.
- **Finish all.** Finishes all debugging sessions.



BPEL Process Instances Window

The BPEL Process Instances window lists all BPEL process instances deployed to the BPEL Service Engine and their currently running instances. For each process instance correlation sets and Faults are listed as subnodes.

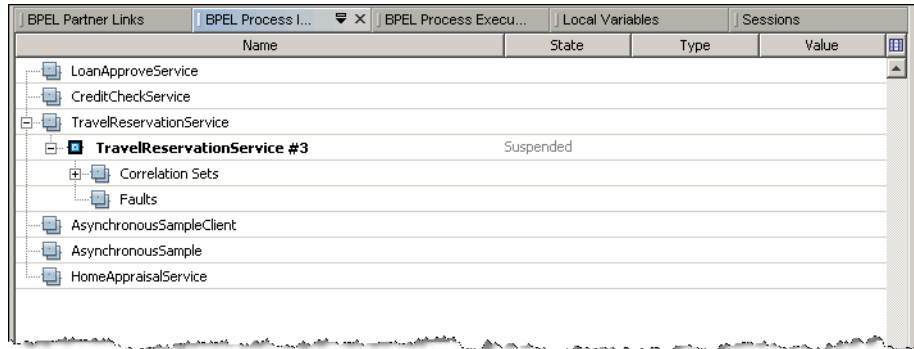
If the current session is not a BPEL Debugger session, this window is empty. The BPEL Process Instances window is populated when a debugging session starts on the BPEL Service Engine, or when the current session is a BPEL Debugger session.

The information displayed for each process instance includes the instance name, unique instance ID, and its state.

Process instances can exist in one of the following states:

- **Running.** The instance is currently being executed on the BPEL Service Engine.
- **Suspended.** The instance has been suspended for some reason. For example, the process instance has reached a breakpoint.
- **Unknown.** The status of the instance is unknown.

A process instance that is current is shown in bold. A process instance becomes current when it reaches a breakpoint or when you manually make it current.



To make a process instance current, do one of the following:

- Double-click the process instance.
- Right-click the process instance and choose Make Current from the pop-up menu.

To terminate a process instance:

- Right-click the process instance and select Terminate from the pop-up menu. The process instance is terminated and removed from the list.

Correlation Sets and Faults information

For the Correlation Sets node the information comes out during the process execution.

For each process instance correlation set, a list of properties it includes is shown. For the properties, type and value information is displayed. Find more information on Correlation sets, properties, and property aliases here: [“Using Correlation” on page 109](#).

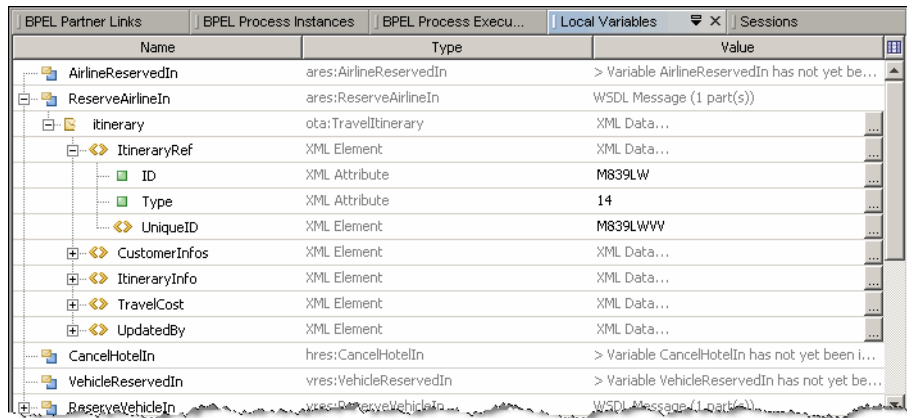
Local Variables Window

The Local Variables window shows the structure of local variables and their values for the current process instance and current position. The current position is the place where the current process instance became suspended. When you change the current process instance, the records in the Local Variables window are updated to reflect the variables for the new current process instance and the new current position.

The structure of local variables is shown as a tree. The information provided for each variable includes the variable name and value.

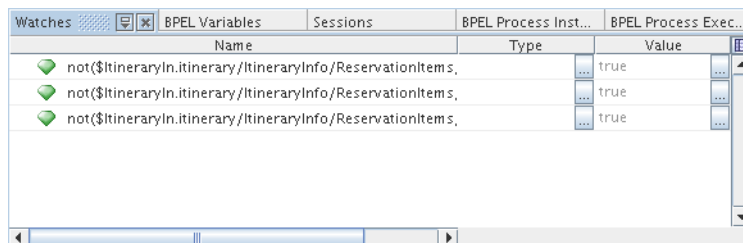
In the Local Variables window you can do the following:

- View the variable structure. To do this, expand the variable node in the tree.
- View and edit the values of variables. To edit the value of a variable, click the ellipsis (...) button and enter the new value in the editor window.
- The structure of local variables is shown as a tree. The information provided for each variable includes the variable name and value.



Watches Window

The Watches window shows the list of XPath expressions that you want to monitor. You add watches explicitly before or during the debugging session. The Watches window shows the expression and its value. The value of the expression may change as the process advances depending on the logic of your process.



▼ To set watches in the BPEL process:

- 1 (Optional) Be sure that the Watches window is visible or choose Window → Debugging → Watches (Alt-Shift-2) to view it.
- 2 If you want to enter an XPath expression from your BPEL process, copy it using one of the following methods:
 - In the Source view, copy the XPath expression you want to watch. The XPath expressions can be found inside the <condition> tag.
 - In the Design view, select an element that has an expression and copy the expression from the Condition row in the Properties window.
- 3 Right-click inside the Watches window and choose New Watch.
- 4 In the Watch Expression field of the New Watch dialog box, do one of the following:
 - Paste the XPath expression you have copied.
 - Enter any valid expression that is compliant with XPath 1.1.
- 5 (Optional) If needed, add more watches.
- 6 Ensure that a debugging session is running and perform a test run.
- 7 As the process instance reaches a breakpoint and becomes suspended, examine the values of the expressions being watched in the Value column of the Watches window.

BPEL Process Execution Window

The BPEL Process Execution window graphically represents the progress of executing the current BPEL process instance in the BPEL Debugger. When you change the current process instance, the process tree in the BPEL Process Execution window is updated to reflect the state of the new current process instance and the new current position.

In the BPEL Process Execution window, the following colors are used to display the state of BPEL activities:

- **Green** — The activity is being executed at the moment.
- **Gray** — The activity has not been executed yet.
- **Black** — The activity has been executed.

BPEL Process Execution Window displays the following information:

- **Name** — The name of the activity.
- **Thread** — The thread in which the activity is/was executed. For the nodes that have not yet been executed no thread information is provided.

- **Line** — Contains the path to the file and the line number for the activity in the file.
- **XPath** — Shows XPath expression pointing to the activity.

Name	Thread	Line	XPath
TravelReservationService		C:\Documents and Settings\ak21770... /bpws:process[1]	
Main	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]	
ReceiveItinerary	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
CopyItineraryIn	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
HasAirline	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
HasVehicle	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
Condition	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
Sequence		C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
HasHotel	CF-2	C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
Condition		C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
Sequence		C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	
ReturnItinerary		C:\Documents and Settings\ak21770... /bpws:process[1]/bpws:sequence[1]...	

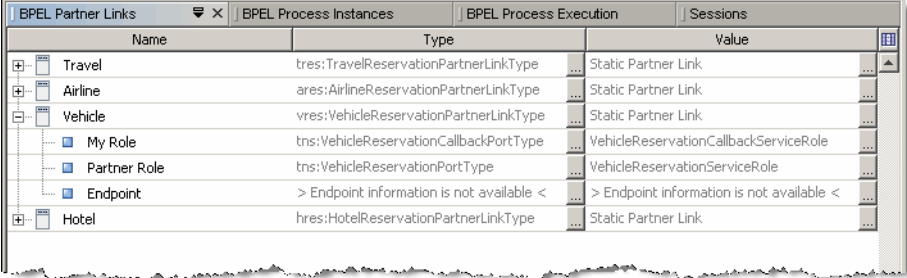
Note – In the BPEL Process Execution window, you can only view the progress of executing BPEL processes. You cannot perform any actions in this window.

BPEL Partner Links Window

The Partner Links window lists all the partner links defined in the BPEL Process.

The information provided for the partner links includes:

- **My Role**
- **Partner Role:** Used for two-way operations only
- **Endpoint:** Endpoint information available for dynamically defined partner links only. For more information see “[Dynamic Partner Links and Dynamic Addressing](#)” on page 63



Name	Type	Value
Travel	tres:TravelReservationPartnerLinkType	Static Partner Link
Airline	ares:AirlineReservationPartnerLinkType	Static Partner Link
Vehicle	vres:VehicleReservationPartnerLinkType	Static Partner Link
My Role	tns:VehicleReservationCallbackPortType	VehicleReservationCallbackServiceRole
Partner Role	tns:VehicleReservationPortType	VehicleReservationServiceRole
Endpoint	> Endpoint information is not available <	> Endpoint information is not available <
Hotel	hres:HotelReservationPartnerLinkType	Static Partner Link

BPEL Debugger Console Messages

You can see the following messages in the BPEL Debugger Console:

Connecting to <host>:<port>

The Debugger is attempting to connect to the BPEL service engine.

Debug session started

The Debugger has successfully connected to the BPEL service engine and the debug session has started.

Unable to start a debug session : Unable to connect to <host>:<port> : Connection timed out: connect

If you see this message, verify the following:

- The GlassFish V2 Application Server is running.
- The BPEL service engine is started.
- The DebugEnabled property of the BPEL service engine is set to true.
- The host name is the host name of the machine that runs the GlassFish V2 Application Server you are connecting to (localhost by default).
- The port value is the same as the DebugPort property of the BPEL service engine you are connecting to (3343 by default).

Unable to start a debug session : Already connected to <host>:<port>

You already have a running debug session attached to this particular service engine.

Debug session terminated : Target disconnected

The Debugger lost connection to the server. Check that the server is running and the network is up.

Stop connecting

You explicitly terminated the debug session when it was connecting.

Debug session finished

You explicitly terminated the debug session when it was running.

Monitoring the BPEL Service Engine

You can monitor BPEL Service Engine instances using the Command Line tool or from your own monitoring application using the BPEL Monitor API library. These monitoring tools enable you to view and change BPEL variable values, and suspend, resume, or terminate BPEL instances.

Installing the BPEL Monitor API and Command Line Monitoring Tool

The `bpelMonitorTool.zip` file contains the BPEL Monitor API, the Command Line Monitoring tool, the BPELManagement API Javadoc and the BPEL Monitor User Manual.

▼ To install the monitoring tool:

- 1 Create a new directory named `bpelsemonitor` in your file-system, for example, `C:\GlassFishESB\glassfish\addons\bpelsemonitor`.
- 2 Download the [bpelMonitorTool.zip](#) and extract the file to your new `bpelsemonitor` directory.
- 3 Edit the `monitorcommand.properties` file, included in the `bpelMonitorTool.zip` file, with appropriate values for your environment.
- 4 Make script executable, as follows:
 - For Unix:
 - a. `cd` to `scripts`.
 - b. `chmod a+x runbpelmonitor.sh`.
 - c. Execute `runbpelmonitor.sh` in the `scripts` directory.
 - For Windows:

Execute `runbpelmonitor.bat` in the `scripts` directory.

Note – If you are using the BPEL Monitor API alone in your own monitoring application, you need the `bpelmonitor-api.jar` and the other JAR files downloaded in the `lib` directory.

Using the BPEL Monitor Command Line Tool

The lightweight command line monitoring tool uses the BPEL Monitoring API library to manage tasks interactively.

▼ To use the BPEL Monitor command line tool

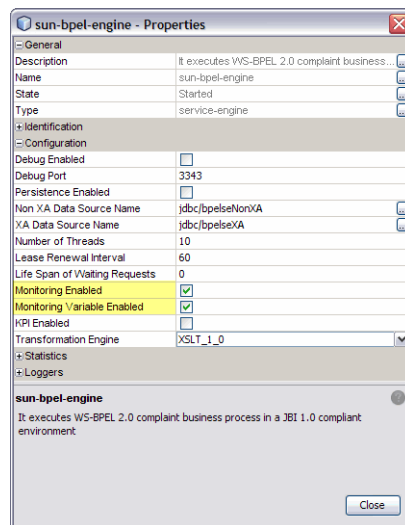
Before You Begin Enable monitoring in the BPEL Service Engine properties.

- 1 From the NetBeans IDE Services window, Start the GlassFish Application Server.
- 2 Start the sun-bpel-engine, under JBI → Service Engines.
- 3 Once the Service Engine has started, open the properties.

There are two properties related to BPEL monitoring:

- **Monitoring Enabled** - required for using the BPEL Monitor API as well as every command in the Command Line Tool for BPEL monitoring.
- **Monitoring Variable Enabled** - required for using variable related API such as `changeVariableValue`, `getVariableValue`, `listBPELVariables`, as well as `getInvokeeInstance`, `getInvokerInstance`, and `command:l, v, k, pr, and ch`.

- 4 Set the Monitoring Enabled property to true (checked). Set Monitoring Variable Enabled to true also if you are using variable related API.



Note – To monitor a BPEL project using the command line tool, deploy the project after the Monitor Enabled property is set to true, otherwise, the "a" command might not produce the correct results.

- 5 From your command line, `cd` to `/bpelmonitor/scripts` and execute `runbpelmonitor.sh`.

The following list of available commands is displayed:

- [a] Show deployed bpel processes
- (su="suName") The service unit name is found in the NetBeans IDE Services window under /JBI/Service Assemblies/YourServiceAssembly/
- [b] Show bpel instances status
([processId="string"] [status="running|completed|suspended|terminated|faulted"]
[instid="instanceId"] [max="maximumInstances"]
[sort="startTime|endTime|updatedAt"] [order="asc|desc"])
- [c] Show activity status on bpel instance
(instid="instanceId")
- [f] Show instance fault
(instid="instanceId")
- [s] Suspend bpel instance
([instid="instanceId1|instanceId2|instanceId3|..."] [processId="string"] [csv="The CSV File"])
- [r] Resume bpel instance
([instid="instanceId1|instanceId2|instanceId3|..."] [processId="string"] [csv="The CSV File"])
- [t] Terminate bpel instance
([instid="instanceId1|instanceId2|instanceId3|..."] [processId="string"] [csv="The CSV File"])
- [l] List bpel variables
(instid="instanceId" [varname="variableName"])
- [v] View bpel variable value
(instid="instanceId" varid="variableId")
- [k] Change bpel variable value
(instid="instanceId" varid="variableId" newval="theNewValue" [part="partName"]
[xpath="theXPathToTheLeafNodeToChange"])
- [ch] Show invokee bpel instances status
(instid="instanceId" [actid="invokeActivityId"])
- [pr] Show invoker bpel instances status
(instid="instanceId" [actid="receiveActivityId"])
- [h] Help
- [e] Exit

6 After the prompt ">", type the letter of the command, or type "help" or "h" for help or examples.

If you type "help" or "h" it displays the list of commands again, and includes the following:

- Press: m + *return* for examples and usage.
- Press: other key + *return* to return.

Command Usage Pattern

The "help" or "h" command displays a list of available commands and the required and optional parameters for each command. For example:

```
[l] List bpeL variables
(instid="instanceId" varname="variableName" )
```

In this example, "l" is the command letter. instid is a required parameter. varname is an optional parameter, which is implied by the enclosing brackets ([]). Parameters that are not enclosed by brackets are required with the command.

To specify a command with parameters do the following:

- After the prompt ">", type the letter of the command and any required parameters and any optional parameters as follows: >l instid="192.168.0.117:757d7361:115d8cdb431:-7fdc" varname="NewWSDLOperationIn"
- Specify the parameter and value as paramName=?Value?. The double quotes are required for the value. There are no spaces between and after '='.
- Type the command and its params in one line, or use '\ ' to continue to next line.
- Specify multiple value for a parm using '|', for example:
>b status="running|completed|terminated"
- You can direct the command output to a file in file system, either in append or overwrite mode. This can be done for any command in the same way, the output to the console is unchanged.

The output can be written to the specified file in the following modes:

- Writing the output to the file, given the full file path, overwriting the file if it currently exists:
>b status="running" >"c:\work\output.txt"
- Writing the output to the file, given the full file path, appending to the file if it currently exists:
>b status="running" >"c:\work\output.txt"+

More Information

For a more extensive manual on how to use the BPEL Monitor Command Line tool, see the BPEL Monitor User Manual included with the ZIP file.

Configuring Quality of Service (QoS) Properties, Throttling, and Redelivery

Quality of Service features are configured from the CASA Editor, and include properties used to configure Retry (Redelivery) and Throttling.

This section contains the following topics:

- [“Configuring the Quality of Service Properties” on page 150](#)
- [“Configuring Message Throttling” on page 155](#)
- [“Configuring Redelivery” on page 155](#)

Configuring the Quality of Service Properties

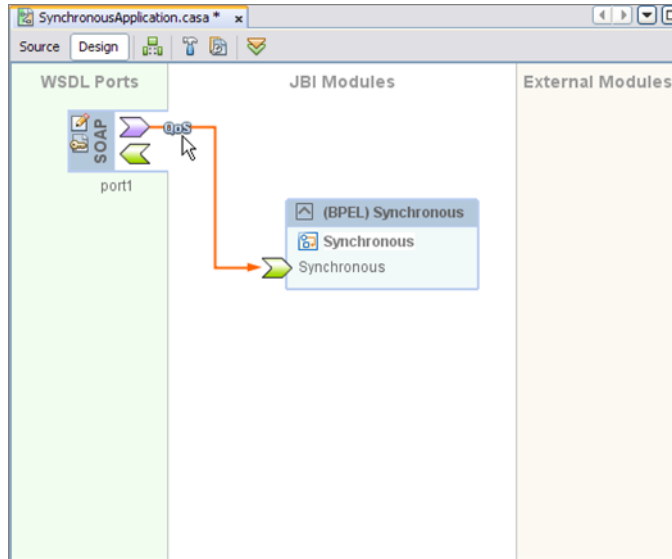
The QoS attributes are configured from the Config QoS Properties Editor, accessed from the Composite Application Service Assembly (CASA) Editor.

For an example of how to access the Config QoS Properties Editor, see [“Configuring an Endpoint for Throttling” on page 155](#)

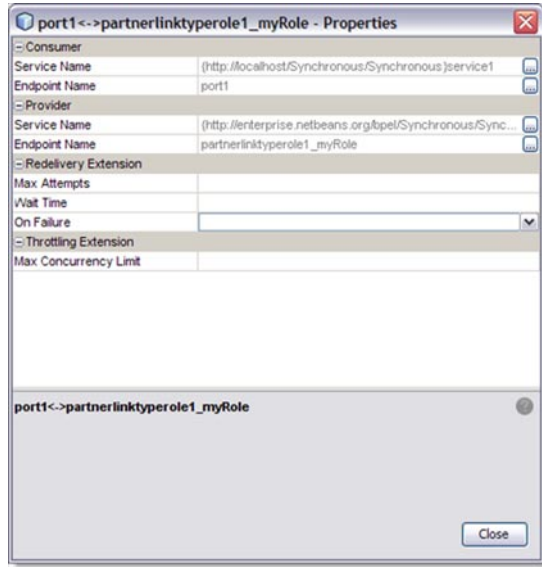
▼ To access the Config QoS Properties Editor

- 1 **From the NetBeans IDE Projects window, right-click the Service Assembly node under your composite application, and select Edit from the pop-up menu.**

The CASA Editor opens containing your composite application.



- 2 If you don't see the connections between your JBI Modules and your WSDL ports, you might need to build your project. Click the CASA Editor's Build Project button.**
All connections are now visible.
- 3 In the CASA Editor, click the QOS icon located on the connection between the JBI Module and the WSDL port you that want to configure.**
The QOS Properties Editor appears.



Quality of Service Properties

Attribute	Description	Value/Example
Consumer Settings		
Service Name	Specifies the consumer service name. Click the ellipses button to open the QName Editor and either select an existing Namespace URI or enter a new Namespace URI and prefix.	{http://j2ee.netbeans.org/wsdl/SoapBasicAuth}SoapBasicAuthPortWssToken
Endpoint Name	Specifies the consumer endpoint name. Click the ellipses button to open an edit window.	SoapBasicAuthPortWssToken
Provider Settings		
Service Name	Specifies the provider service name. Click the ellipses button to open the QName Editor and either select an existing Namespace URI or enter a new Namespace URI and prefix.	{http://enterprise.netbeans.org/bpel/SoapcAuthBP/SoapBasicAuthWssTokenPortType_myRole}SoapBasicAuthWssTokenPortType_myRole
Endpoint Name	Specifies the Provider endpoint name. Click the ellipses button to open an edit window.	SoapBasicAuthWssTokenPortType_myRole
RedeliveryExtension Settings		
maxAttempts	Specifies the number of retries to attempt before using the on-failure option.	20

Attribute	Description	Value/Example
waitTime	Specifies time (in milliseconds) to wait between redelivery attempts.	300

Attribute	Description	Value/Example
on-failure	<p>Specifies the type of action to be taken when message exchange (ME) re-delivery attempts have been exhausted.</p> <p>The on-failure options are</p> <ul style="list-style-type: none"> ■ delete: When the final defined delivery attempt has failed, the QoS utility abandons the message exchanges (ME) and returns a Done status to the JBI component, which proceeds to its next process instance. This option is only supported for <i>In-Only</i> message exchanges. ■ error: When the final defined delivery attempt has failed, the QoS utility returns an Error status to the JBI component, and the JBI component throws an Exception. This is the default option, and is supported for both <i>In-Only</i> and <i>In-Out</i> message exchanges. ■ redirect: Similar to the <i>delete</i> option, except that the QoS utility re-routes the ME to the configured redirect endpoint when the maxAttempts count has been exhausted. If the QoS utility is successful in routing the message to the redirect endpoint, a Done status is returned to the JBI component; otherwise, an Error status is returned. This option is supported for <i>In-Only</i> message exchanges only. ■ suspend: The QoS utility returns an Error status to the JBI component if it is not able to deliver the ME to the actual provisioning endpoint. After the re-delivery attempts have been exhausted, the JBI Component suspends the process instance. This option is only supported if monitoring is enabled in the JBI Component, since the user must use the monitoring tool to resume a suspended instance. This option is supported for both <i>In-Only</i> and <i>In-Out</i> message exchanges. 	delete

Attribute	Description	Value/Example
ThrottlingExtension Settings		
maximum-ConcurrencyLimit	Specifies the maximum number of concurrent messages that can be processed on a specific connection. This number is used to set up the maximum number of concurrent messages that the internal endpoint sends to the the provider endpoint.	10

Configuring Message Throttling

Throttling allows you to set the maximum number of concurrent messages that are processed by a particular endpoint. Increased message load and large message payloads can cause memory usage spikes that can decrease performance. Throttling limits resource consumption so that consistent performance is maintained.

Configuring an Endpoint for Throttling

Throttling is a QOS feature configured from the CASA Editor.

▼ To configure Throttling for an endpoint

- 1 # In the CASA Editor, click the QOS icon located on the link between the JBI Module and the WSDL port you want to configure.

The QOS Properties Editor appears.

- 2 In the QOS Properties Editor, click the property field for maximumConcurrencyLimit under ThrottlingExtension, and enter an integer for the maximum number of concurrent messages allowed for this endpoint.

- 3 Click Close.

The appropriate throttling configuration for the connection is generated in the project's jbi.xml file, when the service assembly is built.

Configuring Redelivery

Redelivery is a Quality of Service mechanism that handles message delivery when first-time delivery fails. Redelivery allows you to define the number of attempts that the system makes to deliver a message, the time between attempts, and the final result for an undeliverable message or non-responsive endpoint.

Redelivery is configured for a specific connection from the Composite Application Service Assembly (CASA) Editor, by clicking the QoS icon for that connection. This opens the Config QoS Properties for that connection. From the RedeliveryExtension section of the editor, configure the Redelivery properties.

The Redelivery configuration parameters are:

- **maxAttempts** — Specifies the number of times that the project attempts to re-deliver a message. An error status is returned to the JBI component for each failed attempt.
- **waitTime** — Specifies the time, in milliseconds, that the project waits between redelivery attempts.
- **on-failure** — Specifies the actions taken and the message destination when the specified redelivery attempts have been exhausted. This parameter has four options: delete, redirect, suspend, and error.

The on-failure parameter has four options: delete, redirect, suspend, and error.

- **delete** — The delete option specifies that when the final attempt to re-deliver the message has failed, the QoS utility deletes the message and returns a Done status to the JBI component, at which time the component proceeds to its next process. The delete option only supports *In-Only* message exchanges.
- **redirect** — The redirect option specifies that after the final attempt to re-deliver the message has failed, the QoS utility redirects the message to a user-defined endpoint, such as a “dead-message” folder. Upon successful delivery to the redirect endpoint, the QoS utility returns a Done status to the JBI component, at which time the component proceeds to its next process. The redirect option only supports *In-Only* message exchanges.
- **suspend** — The suspend option specifies that when the final attempt to re-deliver the message has failed, the JBI component suspends the process instance. This option is only supported if monitoring is enabled in the JBI Component, since the user must use the monitoring tool to resume a suspended instance. This option is supported for both *In-Only* and *In-Out* message exchanges.
- **error** — The error option specifies that when the final attempt to re-deliver the message is exhausted, the JBI component throws an exception. This option is only supported if monitoring is enabled in the JBI Component, since the user must use the monitoring tool to resume a suspended instance. This option is supported for both *In-Only* and *In-Out* message exchanges.

Note: The on-failure options: delete and redirect, cannot be applied to In-Out message exchanges because In-Out message exchanges require a specific response from the process instance to proceed further, and as such, the return value for these options does not suffice.

For more information regarding Redelivery, see [Redelivery \(http://wiki.open-esb.java.net/Wiki.jsp?page=Redelivery\)](http://wiki.open-esb.java.net/Wiki.jsp?page=Redelivery).

Using Dynamic Partner Links and Dynamic Addressing

When you are designing an application, you may need to configure certain services whose endpoints (addresses) are not known beforehand, or it may be necessary to change an endpoint reference while the application is running. The Dynamic Partner link feature allows you to dynamically assign an endpoint reference to the partner link. This means that you can use one partner link for subsequent calls to different web-services (provided that the services use the same interface).

There are several different ways to construct the end point information in BPEL. To deliver the address information to the partner link you can use standard a Assign activity and the BPEL Mapper. In the following examples the endpoint address follows the [WS-Addressing schema](#). Refer to [WS-Addressing standard \(WS-A\)](#).

Note – Dynamic Addressing is only implemented for SOAP (HTTP Binding Component).

Using a Literal to Construct an Endpoint

The BPEL literal syntax can be used to define an endpoint address and assign it to a partner link. The following code sample shows a BPEL literal used to define an HTTP endpoint assigned to a partner link.

```
<bpws:assign name="Assign2">
  <bpws:copy>
    <bpws:from>
      <bpws:literal>
        <sref:service-ref>
          <ns1:EndpointReference>
            <wsa:Address>
              http://localhost:8080/StockQuoteService
/StockQuotePort
            </wsa:Address>
            <wsa:ServiceName PortName="stockQuotePort">
              ns3:stockQuoteService
            </wsa:ServiceName>
          </ns1:EndpointReference>
        </sref:service-ref>
      </bpws:literal>
    </bpws:from>
    bpws:to partnerLink="plStockQuote"/>
  </bpws:copy>
</bpws:assign>
```

In this scenario, you are invoking a partner link that is associated with a SOAP (HTTP) endpoint. That partner link, on the binding side of the application, acts as a proxy for BPEL to the external world.

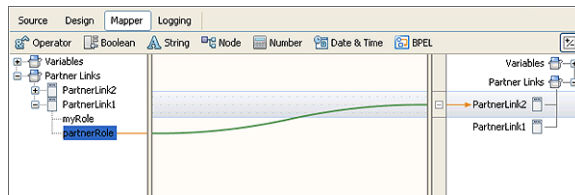
In this example, we have an XML fragment that displays an Assign activity. The Assign has a copy and points to a literal address inline. When you invoke the service, you can assign a different address as a variable value in the invoke properties. In addition to assigning value to variable, you also assign a value to the partner link. Then, when the service is invoked, the HTTP Binding Component stops using the deployment address and instead uses the address that you just assigned.

The partnerLink "plStockQuote" could be subsequently used in an invoke.

Note that the WS-A defined schema object should be wrapped in an element called service-ref, that is defined in the BPEL. For more information about using the service-ref wrapper, refer to the [BPEL Specification, section 6.3](#).

Using an Existing Partner Link's Endpoint

The endpoint of one partner link can be assigned to another partner link. The following BPEL Mapping and code sample show how the partnerRole endpoint of PartnerLink1 is copied to PartnerLink2.



Mapping the PartnerLink1 partnerRole to PartnerLink2 generates the following source code.

```
<assign name="Assign2">
  <copy>
    <from partnerLink="PartnerLink1" endpointReference="partnerRole"/>
    <to partnerLink="PartnerLink2"/>
  </copy>
</assign>
```

In this example, PartnerLink1 is associated with a SOAP address. Instead of using that address, you want to use PartnerLink2. You send the address of PartnerLink1 to PartnerLink2. PartnerLink1 partnerRole endpoint is copied to PartnerLink2.

PartnerLink2 can also be used in subsequent invokes.

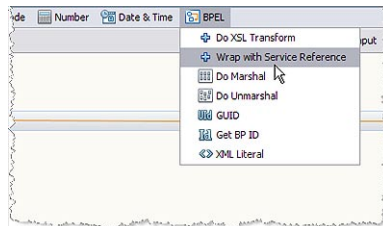
Using an Incoming Message to Extract the Endpoint

To extract an endpoint address from an incoming message, the message must be defined with the endpoint schema as part of the message. The following code sample show one way that this can be done.

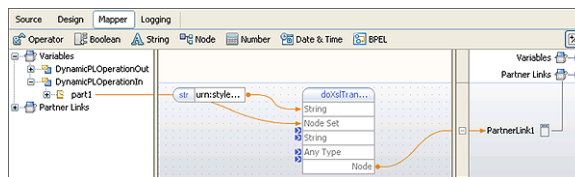
```
<types>
  <xsd:schema targetNamespace="http://j2ee.netbeans.org/wSDL/dynamicPL">
    <xsd:import namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"
      schemaLocation="http://schemas.xmlsoap.org/ws/2004/08/addressing/">
    </xsd:schema>
  </types>
  <message name="dynamicPLOperationRequest">
    <part name="part1" element="wsa:EndpointReference"/>
  </message>
```

Note that the prefix `wsa` is defined to point to `http://schemas.xmlsoap.org/ws/2004/08/addressing`.

Use this message variable to assign the endpoint to a partner link at runtime. A special BPEL mapper function, "Wrap with Service Reference" makes it easy to map the WS-A message to a partner link, as demonstrated below.



Choosing the BPEL Mapper option `Wrap with Service Reference` adds the `doXslTransform` method box to the Mapper canvas.



The doXslTransform Method box generates the BPEL doXslTransform() function, as shown in the code sample below.

```
<assign name="Assign1">
  <copy>
    <from>ns0:doXslTransform('urn:stylesheets:wrap2serviceref.xsl',
$DynamicPLOperationIn.part1)</from>
    <to partnerLink="PartnerLink1"/>
  </copy>
</assign>
```

The stylesheet for the transformation is already defined by the editor.

Using a Database Query to Provide an Endpoint

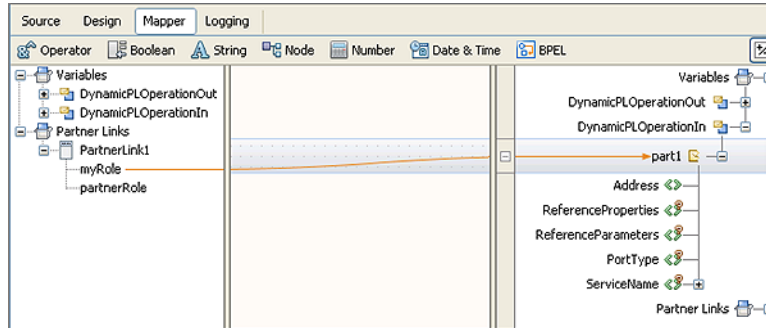
Expanding on the last example, you can also have a BPEL service do a dynamic addressing invocation, with the address coming from a database.

You start with a composite application that is triggered by something such as a input file. You do not want to use the address that is directly assigned to the file, so you do a database query or database select for the address.

The composite application triggers the Database Binding Component to do a simple select from a table that contains addresses. The composite application takes the results from the database query and puts it into the BPEL Mapper and maps the query result to the variable where you normally put your dynamic SOAP address. The Mapper assigns this address to the partner link.

Sending Service Endpoint References

A service can send its own endpoints, that can then be used as call back addresses, using the following type of mapping.



The PartnerLink1 myRole is mapped to the DynamicPLOperationIn part1. This generates the following code sample.

```
<assign name="Assign2">
  <copy>
    <from partnerLink="PartnerLink1" endpointReference="myRole"/>
    <to variable="DynamicPLOperationIn" part="part1"/>
  </copy>
</assign>
```

When the variable “DynamicPLOperationIn” is used in subsequent invokes in the BPEL, the endpoint information is passed on to the partner.

Note – The BPEL Specification states that only the partner address can be changed dynamically. In BPEL terms, this means that only the partnerRole of a partner link can be assigned a value, and myRole does not change after the BPEL process has been deployed.

Configuring Persistence for the BPEL Service Engine

In order to ensure the integrity of your business process data in the case of a system failure, you can configure the BPEL Service Engine to persist business process data to a database. With BPEL Persistence enabled, the BPEL Service Engine recovers business process data that would otherwise be lost, and continues processing from the point of system failure.

The BPEL Service Engine supports MySQL, Derby (JavaDB), and Oracle. This section describes how to configure the BPEL Service Engine, the user, and the database, to persist data.

This section includes the following topics:

- “Setting the JVM Classpath to the Database JDBC Drivers” on page 162
- “Configuring the User and Database for Persistence” on page 163
- “Creating an XA Connection Pool and a JDBC Resource” on page 165
- “Creating a Non-XA Connection Pool and JDBC Resource” on page 167

- [“Enabling Persistence for the BPEL Service Engine” on page 167](#)
- [“Truncating and Dropping Tables” on page 168](#)

Setting the JVM Classpath to the Database JDBC Drivers

The database JDBC drivers must be either set in the GlassFish JVM classpath (preferred) or placed in the `/glassfish/lib` directory before you create the connection pools. This is not necessary for the Derby (JavaDB) database, but is required for Oracle or MySQL. For example, if you are using JDK 6, you would set your GlassFish JVM classpath to the `ojdbc6.jar` file, which is the Oracle database JDBC driver file for JVM 6.

▼ To set the GlassFish JVM Classpath settings

- 1 **Open the GlassFish Admin Console. To access the Admin Console, do the following:**
 - a. From the Services window of the NetBeans IDE, start the GlassFish server.
 - b. Right-click the GlassFish server and select View Admin Console from the popup menu.
The login screen of the Admin Console appears.
 - c. Open the Admin Console using the default username and password:
 - Username: admin
 - Password: adminadmin
- 2 **To access the GlassFish JVM classpath settings:**
 - a. From the Admin Console's Common Tasks window, click Application Server.
The Application Server settings appear in the console window to the right.
 - b. Click the JVM Settings tab and the Path Settings sub-tab.
- 3 **In the Classpath Suffix field, type the full path for your database JDBC driver on your local directory. For example, `C:\GlassFishESB\drivers\oracle11gDriver\ojdbc6.jar`.**
- 4 **Click Save, and restart GlassFish.**

Setting the JVM Classpath for GlassFish Clustering

Additional steps must be taken when setting the GlassFish JVM Classpath to the Database drivers in a clustered environment. For additional information see [“Configuring the BPEL Service Engine for Clustering” in *Configuring GlassFish ESB for Clustering*](#).

Configuring the User and Database for Persistence

This section describes how to configure an appropriate user and database, to persist data.

Derby (JavaDB)

The BPEL Service Engine uses the JDBC resource created in the Application Server to make the necessary database connection for persistence. NetBeans is bundled with the JavaDB database, which is the Sun's supported distribution of the open source Apache Derby database. The BPEL Service Engine is configured to connect to the JavaDB database by default.

It is not necessary to create a user and database for Derby. Just create the connection pools and include the "create" flag in the database name, and set the value to "true". For example:

DatabaseName → bpelseDB;create=true.

Oracle

Create the Oracle database user with required privileges and tablespace for BPEL Service Engine persistence.

▼ To create the Oracle database user

- 1 **Log into Oracle as sysdba. For example, from the command prompt enter: sqlplus "sys/syspassword@tnsentry as sysdba". You need to include sqlplus in the path and the tns entry for the Oracle database.**
- 2 **Execute the following script using the default values:**

```
CREATE TABLESPACE bpelsedb DATAFILE 'bpelsedb.dat' SIZE 512M
  REUSE AUTOEXTEND ON NEXT 2048M MAXSIZE UNLIMITED;
CREATE USER bpelse_user IDENTIFIED BY bpelse_user DEFAULT TABLESPACE
  bpelsedb QUOTA UNLIMITED ON bpelsedb TEMPORARY TABLESPACE temp QUOTA 0M ON system;
GRANT CREATE session to bpelse_user;
GRANT CREATE table to bpelse_user;
GRANT CREATE procedure to bpelse_user;
GRANT select on sys.dba_pending_transactions to bpelse_user;
GRANT select on sys.pending_trans$ to bpelse_user;
GRANT select on sys.dba_2pc_pending to bpelse_user;
GRANT execute on sys.dbms_system to bpelse_user;
GRANT select on SYS.dba_2pc_neighbors to bpelse_user;
GRANT force any transaction to bpelse_user;
```

[Click Here](#), for more information and to download the latest scripts for Oracle.

Note – The above code is wrapped for display purposes.

Tip – The user can also connect to the database from NetBeans or other SQL clients by giving the username "sys as sysdba". The password should be the same as that of the system user. You can also change the user, tablespace and datafile name, and size/quota, as per your choice and requirements.

MySQL

Create the MySQL database user with required privileges and database for BPEL Service Engine persistence.

▼ To create the MySQL database user

- 1 **Log into MySQL as root. For example, from the command prompt enter:** `mysql -h host -u root -p.`

You need to include the MySQL bin folder in the path. You can omit the host if you are connecting from the same machine.

- 2 **Execute the following script using the default values:**

```
CREATE DATABASE BPELSE_USER_DB;  
GRANT ALL ON BPELSE_USER_DB.* TO 'BPELSE_USER'@'%' IDENTIFIED BY 'BPELSE_USER';
```

[Click Here](#), for more information and to download the scripts for MySQL.

Setting max_allowed_packet

When a MySQL client or the MySQL server gets a packet that is larger than the max_allowed_packet bytes, it issues a "Packet too large" error and closes the connection. By default this value is configured low. You must increase this value for large messages. Set the value of this parameter to the size of the biggest message you anticipate.

To set max_allowed_packet:

1. Open the "my.ini" file under the MySQL server install directory.
2. Search for the "max_allowed_packet" parameter. If the file does not have it, add the parameter to the file.
3. Set the value as needed. To set the value to 1GB, enter the value as one of the following:
`max_allowed_packet=1073741824`
`max_allowed_packet=1G`
4. Restart the MySQL Server.

For more information see [Setting max_allowed_packet](#) and [Using Options to Set Program Variables](#).

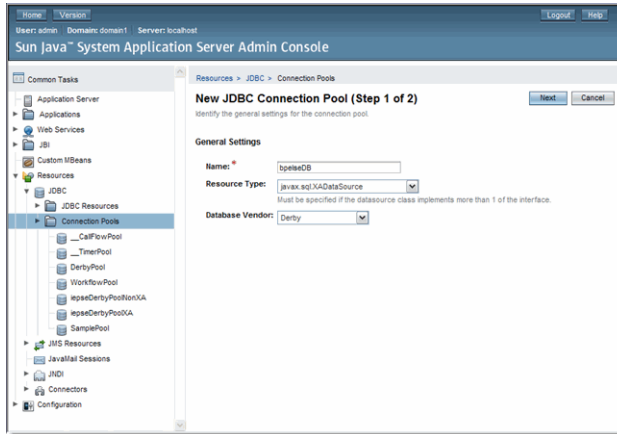
Note – The current version of the JDBC driver for MySQL, MySQL Connector/J, version 5.1.6, has a bug which can cause errors when an XA connection is used. As such, using this driver can cause fatal exceptions in the BPEL Service Engine. A [patch for MySQL Connector/J version 5.1.6](#), is available, and will be included in subsequent releases of MySQL Connector/J. For more information on the patch, see <http://bugs.mysql.com/bug.php?id=35489>.

Creating an XA Connection Pool and a JDBC Resource

Note – Before you setup and test the Connection Pools, you must first set the GlassFish JVM classpath for your database JDBC drivers. See “[Setting the JVM Classpath to the Database JDBC Drivers](#)” on page 162 for more information.

▼ To create an XA Connection Pool:

- 1 Log into the GlassFish Admin Console. To do this, right-click your application server node, in the Services window under Servers, and choose View Admin Console.**
You can also open the Admin Console from your web browser using the correct URL. For example: `http://localhost:4848`.
- 2 In the navigation tree, expand the following nodes: Resources → JDBC.**
- 3 Select Connection Pools, and in the right panel, click the New button.**
- 4 Under General Settings, specify a name (such as bpelseDBXA).**
- 5 Set the Resource Type to `javax.sql.XADataSource`.**
- 6 Set the database vendor to the appropriate database vendor (Derby, Oracle or MySQL). Click next.**



- 7 From the New JDBC Connection Pool (Step 2 of 2) window, under Connection Validation, enable Allow Non Component Callers.
- 8 Under Additional Properties, specify the following properties for your database:
 - **Derby:**
 - ServerName → machine-name
 - DatabaseName → DatabaseName: bpelseDB;create=true
 - User → bpelse_user
 - Password → bpelse_user
 - **Oracle:**
 - URL → jdbc:oracle:thin:@machine-name:port:sid
 - User → bpelse_user
 - Password → bpelse_user
 - **MySQL:**
 - URL → jdbc:mysql://machine-name:port/databasename
 - User → bpelse_user
 - Password → bpelse_user

If you did not use the default values, make sure that the ServerName, User, Password and DatabaseName values are those that you specified when you created the database.

- 9 Click Finish. The new user and database are created.
- 10 Click the connection pool name and click the Ping button to verify your database connection.
- 11 Click Finish.

▼ Create a new JDBC resource:

- 1 From the Admin Console navigation tree, expand the Resources → JDBC nodes, and select JDBC Resources.
- 2 In the right panel, click the New button.
- 3 Provide a JNDIName (such as jdbc/bpelseDB) and specify the JDBC Connection Pool (bpelseDB) you created previously. You will use this JNDIName later when you enable persistence in the BPEL Service Engine properties.
- 4 Expand the Configuration node and select Transaction Service (or Configuration → server-config → Transaction Service, if clustering is configured).
- 5 For the On Restart parameter, enable Automatic Recovery by selecting the Enabled check box.

Creating a Non-XA Connection Pool and JDBC Resource

To create a non-XA Connection Pool, follow the same procedures used to create an XA Connection Pool and JDBC Resource, but substitute bpelseDB as the name, and `javax.sql.ConnectionPoolDataSource` as the Resource Type.

Note – Before you setup and test the Connection Pools, you must first set the GlassFish JVM classpath for your database JDBC drivers. See [“Setting the JVM Classpath to the Database JDBC Drivers” on page 162](#) for more information.

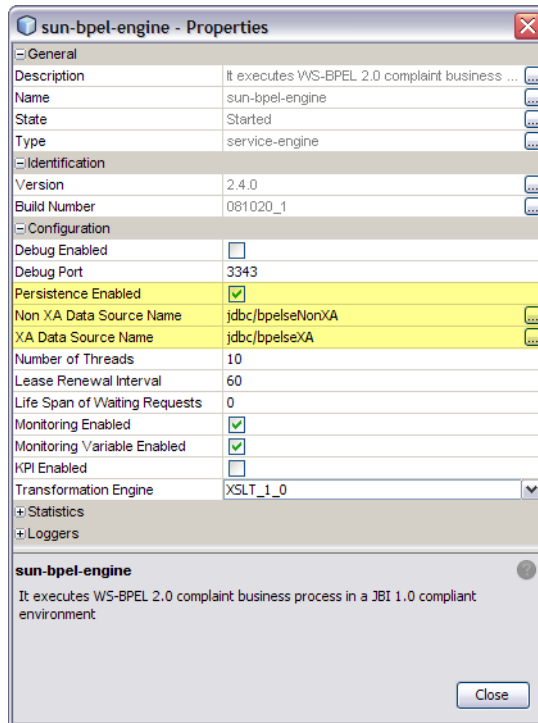
Enabling Persistence for the BPEL Service Engine

Persistence is configured in the BPEL Service Engine runtime properties.

▼ To enable persistence for the BPEL Service Engine

- 1 In the NetBeans IDE Services window, expand the Sun Java System Application Server (GlassFish) → JBI → Service Engines **Nodes**.
- 2 Right-click `sun-bpel-engine` and select **Properties**.
The `sun-bpel-engine` Properties window appears.
- 3 Set the `PersistenceEnabled` property to `true`.

- 4 **Set the Non XA Data Source Name property to specify the JNDIName of the non-XA JDBC resource that you created when you configured the database.**
- 5 **Set the XA Data Source Name property to specify the JNDIName of the XA JDBC resource that you created when you configured the database.**



- 6 **Click Close to save your settings.**
See “Configuring the BPEL Service Engine Runtime Properties” on page 122 for property descriptions.
- 7 **Stop, shut down, and start the BPEL Service Engine to enable your new settings.**

Truncating and Dropping Tables

The following notes provide drop and truncate scripts as well as additional information about configuring persistence. Some of the instructions mentioned here may change, so check back for updates or contact the BPEL Service Engine team if you have questions.

When the BPEL Service Engine is started, it queries the database for the existence of the tables required for persistence. If these tables are not available, the BPEL Service Engine will create the required tables.

Drop and Truncate Scripts

To download the scripts, click [Drop and Truncating Table Scripts](#). You can also refer to [How to Delete From UI](#) for more information.

For the [script to drop an Oracle user](#), use the following:

```
DROP TABLESPACE BPELSE_USER_DB INCLUDING CONTENTS AND DATAFILES CASCADE  
CONSTRAINTS;
```

Configuring Failover for the BPEL Service Engine

In order to optimize and ensure business process throughput on highly scalable systems, the BPEL Service Engine supports load balancing and failover. Load balancing distributes processing over multiple BPEL Service Engines via multiple BPEL service units. Failover prevents processing from being interrupted by picking up business processes from any failed systems and processing them to completion.

Load Balancing

When a business process needs to be scaled to meet heavier processing needs, you can distribute it across multiple service engines, running on multiple processors or systems, to increase throughput. The BPEL Service Engine's load balance algorithm automatically distributes processing across multiple engines.

Failover

When your business process is configured for load balancing, the BPEL Service Engine's failover capabilities ensure throughput of running business process instances. When a business process instance encounters an engine failure, any suspended instances are picked up by the next available BPEL Service Engine.

To configure failover, set the BPEL Service Engine property, `EngineExpiryInterval` to register itself as alive frequently enough to meet the demands of your system. Optimizing this property setting might require some testing. The default setting is 15.

Failover Considerations

In order to configure failover for BPEL Service Engines, you must adhere to the following guidelines:

- Persistence must be enabled for both load balancing and failover.

- To run persistence, all BPEL Service Engines must be restarted.
- Service assemblies must be deployed manually across all clustered JBI environments.
- Failover is implemented consistently for the specific protocol and binding components involved in a given business process.
- Only a single database can be used for all BPEL Service Engines when implementing failover.
- The database must be highly available; should the database fail, failover will fail.
- When a BPEL Service Engine fails, a single BPEL Service Engine picks up those instances without distributing them across the balanced system. Consequently, a large number of failed over instances can overload an entire system, one service engine at a time, as a sort of domino effect.
- All BPEL Service Engines in a load balanced system must reside in the same time zone.

BPEL BluePrints

BPEL BluePrints are developed to teach and promote good practices in developing business processes. Together they present solutions for developing business processes that logically combine, orchestrate, and consume web services.

The BPEL BluePrints are located at:

<https://blueprints.dev.java.net/bpcatalog/ee5/soa/index.html> (<https://blueprints.dev.java.net/bpcatalog/ee5/soa/index.html>)

The following BPEL BluePrints are available for download:

- BPEL BluePrint 1: Synchronous Web Service Interactions Using BPEL
- BPEL BluePrint 2: Asynchronous Web Service Interactions Using BPEL
- BPEL BluePrint 3: Fault Handling Using BPEL
- BPEL BluePrint 4: Message-Based Coordination of Events Using BPEL
- BPEL BluePrint 5: Concurrent Asynchronous Coordination of Events Using BPEL

Troubleshooting

Troubleshooting.

“Using BPEL Schemas Different from the BPEL 2.0 Specification” on page 171

“Ports” on page 172

“Test Run Failures” on page 175

“Disabling Firewalls when Using Servers” on page 175

Using BPEL Schemas Different from the BPEL 2.0 Specification

This release of the BPEL Designer supports the BPEL 2.0 final specification and does not support previous specifications. This means that when you open the BPEL files that comply with the previous versions of the specification, the BPEL Designer shows the Unable to Show the Diagram message.

If you see this message, do the following:

- Check the specification version that the BPEL file complies with. BPEL files that comply with the BPEL 2.0 specification have the following string:


```
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
```

 WSDL files that contain PartnerLinkType definitions should have the following string:


```
xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
```

 Replace the namespaces in your files with those above and try to open the BPEL file in the BPEL Designer.
- Make sure that the BPEL constructs used in your process are compatible with the BPEL 2.0 specification.

Service Endpoint Conflict

When deploying two or more Composite Application projects, a service endpoint conflict might occur and the deployment fails. In case of the service endpoint conflict, the following message is displayed:

```
Deploy service assembly failed. (partial success)
MESSAGE: (SOAPBC_DEPLOY_2) Failed to deploy: java.lang.Exception:
An activated endpoint already has the same SOAP Address location:
http://localhost:18181/SynchronousSample
C:\<...\>\SynchronousSample1Application\nbproject\build-impl.xml:209:
Service assembly deployment failed.
BUILD FAILED (total time: 31 seconds)
```

This could typically arise from trying to deploy nearly identical processes that are packaged in different Composite Application projects. The workaround for this issue is to use different endpoints during the deployment of different Composite Application projects.

Explanation: Even though you are deploying distinct Composite Applications and distinct BPEL processes, by default they will have the same endpoint addresses defined in their SynchronousSample.wsdl files. They will both contain the following endpoint address:

```
<service name="service1">
  <port name="port1" binding="tns:binding1">
    <documentation/>
```

```

        <soap:address location="http://localhost:18181/SynchronousSample"/>
    </port>
</service>

```

If you attempt to deploy two Composite Applications (for example, `SynchronousSampleApplication` and `SynchronousSample1Application`) with identical service endpoints, the deployment of the second one will fail due to the endpoint conflict.

You may wish to deploy more than one version of a Composite Application because you want to modify one or both of these processes and deploy both of them at the same time. Or you may want to compare their behavior. To do this you must first make their endpoint addresses distinct. This means editing the process WSDL file and adjusting the `soap:address location` attribute so that there is no conflict. You can adjust either the port number or the service name. For example, either of these would be sufficiently distinct from the original:

```
<soap:address location="http://localhost:18182/SynchronousSample"/>
```

or

```
<soap:address location="http://localhost:18181/SynchronousSampleNew"/>
```

Relationship of Service Endpoint to Test Cases

Each Test Case in the Composite Application project will attempt to send the input message to the target process when you invoke the Test action. In order to know where to send the message, each test case has a property called `destination`. You can modify this property in the Properties window. To invoke the Properties window, right-click the test case node and choose Properties from the pop-up window.

```
destination=http://localhost:18181/SynchronousSample
```

The value of the `destination` property is set at the time the test case is created. So if you subsequently change the service endpoint you will need to manually adjust the `destination` attribute for any previously generated test cases. Newly generated test cases, of course, will be OK.

Ports

GlassFish V2 Application Server HTTP Port

By default, the installer attempts to configure the Application Server's HTTP port to be 8080. Some of the sample processes assume the 8080 value. If for any reason, the Application Server's HTTP port is not 8080, you will have to make adjustments to the samples.

In particular, the Travel Reservation Service sample will require several adjustments.

Assume, for instance, that the Application Server is listening on HTTP port 8090 (not on the default 8080). In this case, you will have to do the following:

Adjust Reservation Partner Services WSDL files

1. In the TravelReservationService BPEL Module project, change the soap address value in the AirlineReservationService.wsdl from

```
<soap:address
  location="http://localhost:8080/webservice/AirlineReservationService"/>
```

to

```
<soap:address
  location="http://localhost:8090/webservice/AirlineReservationService"/>
```

2. Similarly, update the soap address values in VehicleReservationService.wsdl and HotelReservationService.wsdl.

Note: To find out which HTTP port the Application Server is listening on, open the Services window, right-click the GlassFish V2 Application Server's node and choose View Admin Console. This opens the GlassFish V2 Application Server Administration Console in your browser. Type username and password (default values are admin/adminadmin) and log in. Click Application Server in the left pane and choose the General tab in the right pane. The HTTP port value you need is the first in the **HTTP Port(s):** line.

Alternatively, find the following lines in the Application Server log:

```
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 8080
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 8181
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 4848
```

The value you need is in the first line.

Travel Reservation Service Endpoint Conflict

Refer to the [“Service Endpoint Conflict” on page 171](#) section above for a general description of the problem. In case of the Travel Reservation Service sample, however, you have to take these additional steps:

If port 18181 is not available, and if you want to run TRS on another port, such as port 19191, perform the following steps:

Change URLs

Open TravelReservationService.wsdl.

In the service tag change,

```
soap:address
location="http://localhost:18181/TravelReservation/buildItinerary"/
```

to

```
soap:address
location="http://localhost:19191/TravelReservation/buildItinerary"/
```

Similarly, update URL's for `airlineReserved`, `hotelReserved` and `vehicleReserved`.

Adjust the Partner EJB project, ReservationPartnerServices

Perform the following steps:

1. In the IDE, open the `ReservationPartnerServices` project.
(The IDE created the `ReservationPartnerServices` project in the location where you created the `TravelReservationService` project.)
2. In the Projects window, expand the `ReservationPartnerServices` project node, expand the Configuration Files node, and then double-click the `ejb-jar.xml` node to open the file in the visual editor.
3. In the Design view, under Enterprise Beans, click `ReservationCallbackProviderMDB` to expand the entry. Expand Bean Environment and then Environment Entries.
4. Under Environment Entries, select each entry and click Edit to change the 18181 port number in the Entry Value field.

For example, for `AirlineCallbackURL`, change

```
http://localhost:18181/TravelReservation/airlineReserved
```

to

```
http://localhost:19191/TravelReservation/airlineReserved
```

Update the Destination Property

In the `TravelReservationServiceApplication` Composite project expand the Test node. For each test case node under it:

1. Right-click the test case node and choose Properties.
2. In the Properties window, update the value of the Destination property.

Example:

Change `http://localhost:18181/TravelReservation/buildItinerary`

to

```
http://localhost:19191/TravelReservation/buildItinerary
```

Test Run

When executing a test case:

- If the `Output.xml` file is empty (it is empty after a new test case is created), then you are asked whether the `Output.xml` should be populated with the response from the first test run. This first test run output will indicate that the test run failed.
- If the `Output.xml` file is not empty, then the results obtained are compared with the content of the file; if they match, the test execution is marked as passed.

Test Run Failures

If you receive a failed test run, you can do one of the following:

- Check the response message after a failed test run. The response message is available under the test case node in the Projects window. The response message is time stamped.
You can verify that the response does not match the expected response (that is, `Output.xml`) and this might help you understand the problem.

- Check the Server log file after a failed test run.

To do so, go to the IDE's Runtime tab. Select the `View Server Log` action on the GlassFish V2 Application Server node.

This shows the contents of the server log and might contain information about why a test run failed.

One particular case of test run failures is related to tests that use content-based correlation embedded in `Input.xml` (for example, the `Input.xml` files in the Travel Reservation Service test cases have `<UniqueID>...</UniqueID>` as the basis for correlation). In this situation, if you run the test case when there is already a running process instance initiated by the same test case, the second process instance will not be initiated and the test will fail. The following message will appear in the GlassFish V2 Application Server log:

```
Exception occurred while executing a business process instance.  
com.sun.jbi.engine.bpel.core.bpel.exception.CorrelationAlreadyExists: An instance is associated with the correla  
<...>
```

Disabling Firewalls when Using Servers

You might have to disable any firewall in order to successfully deploy run, debug, or test applications on the Application Server or business processes on the BPEL Server.

Required Correlation Set Usage is Not Detected by the Validation System

The BPEL service engine requires strict usage of correlation sets. Currently the validation system does not detect violations of the following requirements:

- **On Message:** The On Message element must have a valid <correlations> child if the On Message is used in a Pick activity that does not have the createInstance="yes" attribute.
- **Receive:** The Receive element must have a valid <correlations> child if it does not have the createInstance="yes" attribute.
- **On Event:** The On Event element must have a valid <correlations> child.

See the *NetBeans IDE 6.1 Release Notes* for other known issues for the SOA pack.